EK-KA883-TD-PRE

VAX 8800 System Technical Description

Volume 3

FOR INTERNAL USE ONLY

digital™

EK-KA883-TD-PRE

VAX 8800 System Technical Description

Volume 3

FOR INTERNAL USE ONLY

Prepared by Educational Services
of
Digital Equipment Corporation

Preliminary Edition, July 1986

Class A Computing Devices

Notice: This equipment generates, uses, and may emit radio frequency energy. The equipment has been type tested and found to comply with the limits for a Class A computing device pursuant to Subpart J of Part 15 of FCC rules, which are designed to provide reasonable protection against such radio frequency interference when operated in a commercial environment. Operation of this equipment in a residential area may cause interference in which case the user at his own expense may be required to take measures to correct the interference.

CONTENTS

SECTION 9 MEMORY SYSTEM (MBOX)

CHAPTER 1 INTRODUCTION

1.1 MANUAL SCOPE 1-1
1.2 WRITING PHILOSOPHY 1-1
1.3 MBOX FUNCTIONS 1-2
1.4 MBOX OVERVIEW. 1-4
1.4.1 NMI Signals Used by the MBox 1-6
1.4.2 MBox Operations. 1-6
1.4.3 Octaword Sort-of-Write 1-6
1.4.4 Command Bus Cycles 1-6
1.4.4.1 Write Longword Bus Cycles (Table 1-3). 1-9
1.4.4.2 Write Quadword Bus Cycles (Table 1-5). . . . 1-10
1.4.4.3 Write Octaword Bus Cycles (Table 1-6). . . . 1-11
1.4.4.4 Read Longword Bus Cycles (Table 1-7) 1-11
1.4.4.5 Read Octaword Bus Cycles (Table 1-8) 1-13
1.4.4.6 Read Hexword Bus Cycles (Table 1-9). 1-14
1.4.5 Memory Controller (MCL). 1-14
1.4.5.1 Command/Address Sequence 1-16
1.4.5.2 Normal Write 1-20
1.4.5.3 Read 1-22
1.4.5.4 Masked Write 1-25
1.4.6 Four-Megabyte Memory Array Board (MAR4). . . . 1-29
1.4.6.1 MAR4 Select and Command/Address. 1-30
1.4.6.2 Write Operation. 1-32
1.4.6.3 Read Operation 1-33

CHAPTER 2 MEMORY CONTROLLER (MCL)

2.1 DFA OVERVIEW (Figure 2-1). 2-1
2.1.1 Command/Address Cycle. 2-1
2.1.2 Write Data Cycle(s). 2-5
2.1.3 First Read Data Cycle. 2-7
2.1.4 "Next" Read Data Cycles. 2-8
2.1.5 NMI Interrupt. 2-9
2.1.6 NMI Memory Busy. 2-9
2.1.7 Single-Bit Error Correction During a Masked Write 2-10
2.1.8 CSR Reads. 2-10
2.2 DATA/ADDRESS (DAD) MCAS. 2-10
2.2.1 DAD Ports. 2-13
2.2.2 NMI Writes to Memory 2-13
2.2.3 NMI Reads from Memory. 2-13
2.2.4 Masked Writes Requiring Single-Bit Error Correction 2-14
2.2.5 Error-Free Masked Writes 2-15
2.2.6 Decode RAM Addressing. 2-15
2.2.6.1 Initially Loading the Decode RAM 2-15
2.2.6.2 Reading CSR1 2-15
2.2.7 CSR Reads to the NMI 2-16
2.3 FUNCTION (FUNK) MCA. 2-17
2.3.1 Function Field Parity. 2-20

2.3.2 Function Decoder 2-20
2.3.3 NEW CMD EARLY/NEW CMD LATE 2-21
2.3.4 Read Lock Function 2-22
2.3.5 Write Unlock Function. 2-22
2.3.6 Lock-Timeout Counter 2-23
2.3.7 Block Command. 2-23
2.3.8 Write Sequence Fault 2-24
2.3.8.1 Write Longword to Memory 2-24
2.3.8.2 Write Longword to CSR. 2-26
2.3.8.3 Write Quadword 2-26
2.3.8.4 Write Octaword 2-27
2.3.9 NMI Faults 2-28
2.3.10 NMI Confirmation 2-30
2.3.11 NMI DEAD 2-31
2.3.12 CSRs (Figure 2-4). 2-32
2.3.13 Read/Return and Read/Continue (Figure 2-5) . . 2-32
2.3.13.1 MCL Immediately Gets the NMI 2-33
2.3.13.2 MCL Waits for the NMI. 2-34
2.3.14 MRM Hold Command (Figure 2-6). 2-35
2.3.15 Force One Cycle (Figure 2-6) 2-35
2.4 ARBITRATION/ID (ARID) MCA. 2-35
2.4.1 NMI Data Parity (Figure 2-7) 2-35
2.4.1.1 Parity In. 2-34
2.4.1.2 Parity Out 2-37
2.4.2 NMI Function/ID Parity (Figure 2-7). 2-38
2.4.2.1 Parity In. 2-38
2.4.2.2 Parity Out 2-38
2.4.3 Fault Detect (Figure 2-8). 2-38
2.4.4 NMI ID/Mask (Figure 2-9) 2-40
2.4.4.1 ID/Mask In 2-40
2.4.4.2 ID Out 2-42
2.4.5 Arbitration/Hold Logic 2-43
2.4.5.1 Memory Gets the Bus Right Away - Longword Read. 2-43
2.4.5.2 Memory Gets the Bus Right Away - Octaword Read (Figure 2-11). 2-45
2.4.5.3 Memory Gets the Bus Right Away - Longword Read Back-to-Back with Another Read Function. 2-45
2.4.5.4 Memory Gets the Bus Right Away - Hexword Read 2-46
2.4.5.5 Memory Does Not Get the Bus Right Away - Longword Read. 2-46
2.4.5.6 Memory Does not Get the Bus Right Away - Two Longword Reads Back-to-Back or an Octaword Read. 2-46
2.4.6 Interrupts (Figure 2-12) 2-47
2.4.7 CSRs (Figure 2-13) 2-49
2.4.7.1 CSR0 2-49
2.4.7.2 CSR3 2-49
2.4.8 Memory Busy (Figure 2-14). 2-49
2.4.9 Clocks and Clock Control (Figure 2-15) 2-50
2.5 MDP OVERVIEW (Figure 2-16) 2-54
2.5.1 MDB Address In 2-54

2.5.2 MDB Address Out. 2-54
2.5.3 MDB Data In. 2-54
2.5.4 MDB Data Out 2-56
2.5.5 Write-Enable and Bad-Data Bits 2-56
2.5.6 Data Parity. 2-57
2.5.7 Data Read Operation. 2-57
2.5.8 Masked Write Operation 2-58
2.5.9 CSR Reads. 2-59
2.6 MEMORY DATA BUFFER (MDB) (Figure 2-17) 2-59
2.6.1 Address Read Port. 2-61
2.6.2 Data Read Port 2-62
2.6.3 W Write Port 2-62
2.6.4 C Write Port 2-63
2.6.5 CSR Logic. 2-64
2.7 MEMORY DATA BUFFER CONTROL (MDBC) MCA. 2-66
2.7.1 Loading Data into the MDB. 2-67
2.7.1.1 Normal Octaword Write With X and Y Buffers Empty. 2-67
2.7.1.2 Normal Octaword Write With Data Already in the X or Y Buffer 2-74
2.7.1.3 Masked Write With No Errors. 2-76
2.7.1.4 Masked Write With a Correctable Error. . . . 2-78
2.7.1.5 Masked Write With an Uncorrectable Error . . 2-79
2.7.2 Unloading Data From the MDB. 2-80
2.7.2.1 General. 2-80
2.7.2.2 Detailed 2-82
2.7.3 Y Out Select Logic 2-83
2.7.3.1 General. 2-83
2.7.3.2 Detailed 2-84
2.7.4 Internal Error, Write Decode RAM, and Clocks 2-87
2.8 DATA CHECK (DCHK) MCA. 2-87
2.8.1 Syndrome Generate. 2-89
2.8.2 Error Check. 2-89
2.8.3 Error Status 2-92
2.8.4 Serializer and CSR2. 2-95
2.8.5 Diagnostic Mode (Figure 2-28). 2-95
2.8.6 Reset and Clocks (Figure 2-28) 2-98
2.9 MRM OVERVIEW (Figure 2-33) 2-99
2.9.1 NAB Board Select 2-104
2.9.2 NAB Command Field and Parity 2-105
2.9.3 MDB Address Selection. 2-105
2.9.4 Internal Error 2-106
2.9.5 MDB Write-Data Load. 2-106
2.9.6 Octaword Writes. 2-106
2.9.7 Read-Data Cycle(s) 2-106
2.9.8 Masked Writes. 2-107
2.9.9 CSR READS. 2-108
2.10 MEMORY SEQUENCE CONTROL (MSC) MCA) 2-108
2.10.1 MSC Buffer Control (Figure 2-35) 2-109
2.10.1.1 Buffer Control Operation 2-109
2.10.1.2 AMRM BUSY REQ. 2-112
2.10.1.3 A CMD PROC START 2-113

2.10.2 BNUM Probe Buffer and Error Logic (Figure 2-36). 2-113
2.10.2.1 Probe Logic. 2-113
2.10.2.2 Error Logic. 2-116
2.10.3 Command/Address/Size Buffer (Figure 2-37). . . 2-116
2.10.3.1 Command Channel. 2-116
2.10.3.2 Address/Size Channel 2-118
2.10.4 Size Logic (Figure 2-38) 2-118
2.10.5 Starting Address Logic (Figure 2-40) 2-120
2.10.5.1 Initial Starting Address 2-120
2.10.5.2 Address Incrementation 2-124
2.10.6 Mask Address/Size Buffer (Figure 2-41) 2-124
2.10.7 Write Command Logic (Figure 2-42). 2-126
2.10.7.1 Write Machine. 2-126
2.10.7.2 Write Command Bits 2-128
2.10.8 Masked-Write Logic (Figure 2-43) 2-129
2.10.9 Command Done Logic (Figure 2-44) 2-131
2.10.9.1 BMSC PRE CMD DONE. 2-131
2.10.9.2 BMSC PRE MASK DONE 2-133
2.10.10 Command Parity 2-133
2.11 MEMORY SEQUENCE CONTROL 1 (MSC1) MCA 2-133
2.11.1 Mask Store (Figure 2-45) 2-134
2.11.2 Select Out Buffer Control (Figure 2-46). . . . 2-137
2.11.3 Read Buffer Control (Figure 2-47). 2-139
2.11.4 MDB Address I/O Select Logic (Figure 2-48) . . 2-139
2.11.4.1 MDB Address In Select Bits 2-139
2.11.4.2 MDB Address Out Select Bits. 2-142
2.11.5 Error Address Pointer. 2-143
2.11.6 BMRM INVERT ADDR4. 2-143
2.12 MEMORY ARRAY SEQUENCE CONTROL (MASC) MCA (Figure 2-49). 2-143
2.12.1 Command/Address Parity 2-143
2.12.2 Force Parity Error 2-145
2.12.3 Board Number (BMAS BNUM<2:0>). 2-145
2.12.4 Send No Command. 2-145
2.12.5 MASC Empty 2-146
2.12.6 Board Select (BMAS BD SEL<2:0>). 2-146
2.12.7 Command Accept (BMAS CMD ACPT) and Board Valid (BMAS BD VALID). 2-146
2.13 READ CONTROL SEQUENCER (RCS) MCA 2-147
2.13.1 Power Control (Figure 2-50). 2-147
2.13.2 CSR Control (Figure 2-51). 2-149
2.13.2.1 BMRM EN SERIAL RD. 2-149
2.13.2.2 BMRM SERIAL RD<2:0>. 2-149
2.13.2.3 AMRM CSR WRITE 2-149
2.13.2.4 ARCS FORCE CMD ACPT. 2-151
2.13.2.5 AMRM MPR DATA SEL. 2-151
2.13.2.6 BMRM FAKE CMD ACPT 2-151
2.13.3 Read Command Bits (Figure 2-52). 2-151
2.13.3.1 AMRM READ CMD<0> 2-151
2.13.3.2 BRCS READ CMD<1> 2-151
2.13.4 Board Select/Enable (Figure 2-52). 2-151
2.13.4.1 Board Select 2-151
2.13.4.2 Board Select Enable. 2-153

2.13.5 Read Data In Signals (Figure 2-52) 2-153
2.13.5.1 AMRM DRIVE NEW DATA. 2-153
2.13.5.2 BMRM NAB GATE. 2-153
2.13.6 RCS Full/Empty Status (Figure 2-52). 2-154
2.13.6.1 ARCS FULL. 2-154
2.13.6.2 BRCS EMPTY 2-154
2.14 BATTERY BACKUP UNIT (BBU). 2-154
2.14.1 Loss of Power. 2-156
2.14.2 Return of Power. 2-156

CHAPTER 3 FOUR MEGABYTE MEMORY ARRAY BOARD (MAR4)

3.1 VAX 8800 ARRAY BUS (NAB) 3-1
3.1.1 Signal Clocks. 3-1
3.1.2 Longword Write Timing. 3-1
3.1.3 Longword Read Timing 3-8
3.1.4 Octaword Read Timing 3-8
3.2 MAR4 OVERVIEW. 3-10
3.2.1 Write Operation (Figures 3-4 and 3-5). 3-12
3.2.2 Read Operation (Figures 3-4 and 3-6) 3-19
3.3 MAR4 DETAILED DESCRIPTIONS 3-22
3.3.1 Clock Logic. 3-23
3.3.2 4MBARRAY Banks 3-26
3.3.2.1 Array Bank Components. 3-26
3.3.2.2 Data Flow. 3-29
3.3.2.3 Array Command Sequencing 3-29
3.3.2.4 Array Refresh Sequencing 3-34
3.3.2.5 Battery Mode 3-36
3.3.2.6 Cold Start 3-38
3.3.2.7 Array Bank Differences 3-38
3.3.3 Input Parser 3-38
3.3.3.1 Command/Address Parity Check 3-39
3.3.3.2 Write Inhibit. 3-39
3.3.3.3 Data Ready Done. 3-39
3.3.3.4 MAR4 Board Selection 3-41
3.3.3.5 Generation of Array Bank Signals 3-41
3.3.3.6 Array Not Busy 3-42
3.3.3.7 Battery Mode 3-42
3.3.4 ECC/DPARITY. 3-42
3.3.4.1 ECC Check Bits 3-42
3.3.4.2 Write Inhibit. 3-42
3.3.4.3 Write Data Parity Check. 3-44
3.3.4.4 INT BAD DATA 3-44
3.3.5 Data Output Control. 3-44
3.3.5.1 MAR4 Read Enable 3-44
3.3.5.2 Bank Select. 3-46
3.3.5.3 MAR4 Data Transfer Enable. 3-47
3.3.5.4 Control of Read Data Transfer. 3-47
3.3.5.5 DRDY SNC CLK 3-48
3.3.5.6 Battery Mode 3-48
3.3.6 Refresh. 3-48
3.3.6.1 Normal Mode. 3-49
3.3.6.2 Battery Mode 3-53

FIGURES

No.	Title	Page
1-1	MBox Simplified Block Diagram.	1-3
1-2	MBox Read/Write Simplified Block Diagram	1-5
1-3	MBox Block Diagram	1-17
1-4	Command/Address Flow Diagram	1-18
1-5	Write Data Cycly Flow Diagram.	1-21
1-6	Read Data Cycle Flow Diagram	1-23
1-7	Masked Write Data Cycle Flow Diagram	1-27
1-8	MAR4 Read/Write Flow Diagram	1-31
1-9	MAR4 Command Fields.	1-32
2-1	DFA Block Diagram.	2-3
2-2	DAD Block Diagram.	2-11
2-3	FUNK Function and Control Logic.	2-18
2-4	FUNK CSRs.	2-25
2-5	Read/Return and Read/Continue Logic.	2-29
2-6	Clock and Command Control Logic.	2-36
2-7	Parity Generation and Checking	2-39
2-8	Fault Detect Logic	2-40
2-9	ID/Mask Logic.	2-41
2-10	Arbitration/Hold Logic	2-44
2-11	NMI Arbitration/Hold Timing.	2-45
2-12	Interrupt Logic.	2-48
2-13	CSR Logic.	2-51
2-14	Memory Busy Logic.	2-52
2-15	Clock, Reset, and Unjam Logic.	2-53
2-16	Memory Data Path (MDP) Block Diagram	2-55
2-17	Memory Data Buffer (MDB) Block Diagram	2-60
2-18	CSR Logic.	2-65
2-19	MDBC -- MDB Data-In Selection.	2-69
2-20	Input Load Command Detect Logic.	2-70
2-21	Full Logic	2-71
2-22	X and Y Bit Storage.	2-72
2-23	MDBC -- MDB Feedback Selection	2-77
2-24	Double-Bit Error Logic	2-80
2-25	MDBC -- MDB Data-Out Selection	2-81
2-26	Y Out Select Flow Diagram.	2-85
2-27	MDBC -- Internal Error, Write Decode RAM, and Clocks	2-88
2-28	DCHK Block Diagram	2-90
2-29	Error Check Block Diagram.	2-91
2-30	Error Status Block Diagram	2-93
2-31	Serializer Block Diagram	2-96
2-32	CSR2 Bit Map	2-97
2-33	MRM Block Diagram.	2-100
2-34	MSC Block Diagram.	2-110
2-35	Buffer Control	2-111
2-36	BNUM Probe Buffer and Error Logic.	2-114
2-37	Command/Address/Size Buffer.	2-117
2-38	Size Logic	2-119
2-39	Hex State Machine Flow Diagram	2-121

2-40 Starting Address Logic 2-122
2-41 Mask Address/Size Buffer 2-125
2-42 Write Command Logic. 2-127
2-43 Mask Write Logic 2-130
2-44 Command Done Logic 2-132
2-45 Mask Store Block Diagram 2-135
2-46 Select-Out Buffer Control Block Diagram. 2-138
2-47 Read Buffer Control Block Diagram. 2-140
2-48 MDB Address I/O Select Block Diagram 2-141
2-49 MASC Block Diagram 2-144
2-50 Power Control. 2-148
2-51 CSR Control. 2-150
2-52 Array Read Control 2-152
2-53 MCL BBU Block Diagram. 2-155
2-54 Power Down Flow Diagram. 2-157
2-55 Power Up Flow Diagram. 2-158

3-1 Longword Write Timing Diagram. 3-5
3-2 Longword Read Timing Diagram 3-6
3-3 Octaword Read Timing Diagram 3-7
3-4 MAR4 Block Diagram 3-11
3-5 Write Flow Diagram 3-13
3-6 Read Flow Diagram. 3-15
3-7 Clock Logic Block Diagram. 3-24
3-8 Clock Timing Diagram 3-25
3-9 4MBARRAY Bank Block Diagram (Bank 0 Shown) . . . 3-27
3-10 Array Command Flow Diagram 3-30
3-11 Array Refresh Flow Diagram 3-35
3-12 Input Parser Block Diagram 3-40
3-13 ECC/DPARITY Block Diagram. 3-43
3-14 Data Output Control Block Diagram. 3-45
3-15 Refresh Time Periods 3-50
3-16 Refresh Block Diagram. 3-51
3-17 Refresh Flow Diagram -- Normal Mode. 3-52
3-18 Refresh Flow Diagram -- Battery Mode 3-55
3-19 Initiation of Battery Mode Refreshes 3-57
3-20 Termination of Battery Mode Refreshes. 3-58

A-1 Flow-Diagram Symbols A-1

TABLES

No. Title Page

1-1 NMI Signals Used by the MBox 1-7
1-2 MBox Command Functions 1-8
1-3 Write Longword Bus Cycles. 1-9
1-4 NMI Confirmation Codes 1-10
1-5 Write Quadword* Bus Cycles 1-10
1-6 Write Octaword Bus Cycles. 1-11
1-7 Read Longword Bus Cycles 1-12
1-8 Read Octaword Bus Cycles 1-13
1-9 Read Hexword Bus Cycles. 1-15

2-1 Command Code 2-2
2-2 Size Code. 2-2
2-3 Function Codes 2-20
2-4 NMI Confirmation Codes 2-30
2-5 Read Function Codes. 2-34
2-6 Read Command Code. 2-59
2-7 Write Commands 2-82
2-8 ENABLE ECC Truth Table 2-99
2-9 Size Code. 2-118
2-10 Initial Address Truth Table. 2-123
2-11 Write State Code 2-128
2-12 Write Command Code 2-128

3-1 NAB Signals. 3-2
3-2 Clock Distribution 3-26
3-3 Read Bank Shift Register Modes 2-46

SECTION 10 NBI (NMI TO VAXBI ADAPTER)

CHAPTER 1 INTRODUCTION

1.1 GENERAL INFORMATION. 1-1
1.2 PHYSICAL DESCRIPTION AND CIRCUIT TECHNOLOGY. 1-3
1.3 BASIC BLOCK DIAGRAM. 1-3
1.4 BASIC OPERATION. 1-7
1.4.1 CPU Read/Write Data Transfers. 1-7
1.4.2 DMA Read/Write Data Transfers. 1-10
1.4.3 Interrupt Operation. 1-13
1.5 NBI REGISTERS. 1-17
1.5.1 NBIA Registers 1-17
1.5.1.1 Control/Status Registers (CSR0 and CSR1) . . 1-18
1.5.1.2 Vector Registers (BR4VR through BR7VR) . . . 1-24
1.5.2 NBIB (BIIC) Registers. 1-25
1.5.2.1 Device Register (DTYPE). 1-26
1.5.2.2 VAXBI Control/Status Register (BICSR). . . . 1-27
1.5.2.3 Bus Error Register (BER) 1-30
1.5.2.4 Error Interrupt Control Register (EINTRCSR) 1-34
1.5.2.5 INTR Destination Register (INTRDES). 1-36
1.5.2.6 IPINTR Mask Register (IPINTRMSK) 1-37
1.5.2.7 IPINTR/STOP Destination Register (FIPSDES). 1-38
1.5.2.8 IPINTR Source Register (IPINTRSRC) 1-39
1.5.2.9 Starting and Ending Address Registers (SADR and EADR). 1-40
1.5.2.10 BCI Control and Status Register (BCICSR) . . 1-42
1.5.2.11 Write Status Register (WSTAT). 1-45
1.5.2.12 Force IPINTR/STOP Command Register (FIPSCMD). 1-46
1.5.2.13 User Interrupt Control Register (UINTRCSR) 1-47
1.5.2.14 General Purpose Registers (GPR <3:0>). . . . 1-49

CHAPTER 2 INTERFACE DESCRIPTIONS

2.1 NMI. 2-1
2.1.1 NMI Signals. 2-4
2.1.2 Basic Timing 2-4
2.1.3 NMI Address Space. 2-11
2.1.4 NMI Read/Write Transactions. 2-11
2.1.5 NMI Arbitration/Memory Busy. 2-16
2.1.6 NMI Interrupts 2-20
2.1.7 NMI Errors 2-23
2.2 DATA BUS (BETWEEN NBIA AND NBIB) 2-24
2.3 VAXBI. 2-29
2.3.1 VAXBI Signals. 2-29
2.3.2 Basic Timing 2-34
2.3.3 VAXBI Address Space. 2-35
2.3.4 VAXBI Read/Write Transactions. 2-40
2.3.5 Interrupt Operation (INTR, IDENT, and IPINTR Transactions). 2-44
2.3.5.1 Interrupt (INTR) Transactions. 2-45
2.3.5.2 Identify (IDENT) Transactions. 2-47
2.3.5.3 Interprocessor Interrupt (IPINTR) Transactions 2-50
2.3.6 STOP Transactions. 2-52
2.3.7 Invalidate (INVAL) Transactions. 2-54
2.3.8 Bus Arbitration. 2-56
2.3.8.1 Bus Requests 2-56
2.3.8.2 Arbitration Modes. 2-57
2.3.8.3 Arbitration Control. 2-57
2.3.8.4 Extending a Transaction. 2-59
2.3.8.5 Special Mode Functions 2-59
2.3.9 VAXBI Errors 2-60
2.3.9.1 Parity Checking. 2-61
2.3.9.2 Transmit Check Error Detection 2-61
2.3.9.3 Protocol Checking. 2-62

CHAPTER 3 FUNCTIONAL DESCRIPTION

3.1 INTRODUCTION 3-1
3.1.1 NBIA Block Diagram 3-1
3.1.1.1 NMI Data Buffer. 3-2
3.1.1.2 NPAR MCA 3-4
3.1.1.3 NBIM MCA 3-4
3.1.1.4 NBFD MCA 3-4
3.1.1.5 NBAP MCA 3-4
3.1.1.6 NBCT MCA 3-5
3.1.1.7 DSEQ MCA 3-5
3.1.1.8 DC022 Transaction Buffer 3-5
3.1.1.9 Data (Bus) Buffers 3-5
3.1.1.10 Data Bus (and Transaction Buffer) Controls . . . 3-5
3.1.2 NBIB Block Diagram 3-6
3.1.2.1 Data Bus Data Buffer 3-6
3.1.2.2 BCI Data Buffer. 3-6
3.1.2.3 Parity and Translation Logic 3-6

3.1.2.4 Data Buffer Read/Write Control 3-8
3.1.2.5 Length and Interrupt Control Logic 3-8
3.1.2.6 Master and Slave Port Sequencers 3-8
3.1.2.7 BIIC 3-8
3.1.2.8 VAXBI Clock Driver/Receiver. 3-9
3.2 INITIALIZATION/SELFTEST. 3-12
3.2.1 Basic NBI Initialization 3-12
3.2.2 BIIC Initialization/Selftest 3-14
3.2.3 Powerup. 3-14
3.2.4 NBI INIT/UNJAM 3-16
3.2.5 RESET (By Connected VAXBI Device). 3-18
3.3 CPU READ/WRITE OPERATIONS. 3-20
3.3.1 NMI Address Decoding and Translation 3-22
3.3.2 Local Read/Write Operations. 3-24
3.3.2.1 Command/Address Cycle. 3-25
3.3.2.2 Write Data Cycle 3-28
3.3.2.3 Return Data Cycle. 3-30
3.3.2.4 Parity Generation and Checking 3-32
3.3.3 VAXBI Read/Write (and IDENT) Operations. . . . 3-33
3.3.3.1 Command/Address Transfer 3-36
3.3.3.2 Write Data Transfer. 3-43
3.3.3.3 Return Read Data Transfer. 3-49
3.3.3.4 Parity Generation and Checking 3-55
3.3.4 Write Sequence Faults. 3-58
3.3.5 NMI BUS ACCESS TIMEOUTS. 3-58
3.3.6 VAXBI Errors 3-58
3.4 DMA READ/WRITE OPERATIONS. 3-60
3.4.1 Command/Address Transfer 3-66
3.4.1.1 Command/Address to BCI Data Buffer and Data Bus Buffer. 3-66
3.4.1.2 Command/Address to NBIA's Transaction Buffer 3-70
3.4.1.3 Command/Address to NMI (NMI Command/Address Cycle) 3-70
3.4.2 Write Data Transfer. 3-72
3.4.2.1 Write Data to BCI Data Buffer and Data Bus Buffer. 3-73
3.4.2.2 End of VAXBI Transaction (and Retries) . . . 3-76
3.4.2.3 Write Data to NBIA's Transaction Buffer. . . 3-76
3.4.2.4 Write Data to NMI (NMI Write Data Cycle or Cycles) 3-77
3.4.2.5 NMI Write Transaction Retries (NO ACCESS/MEMORY BUSY/NOACK). 3-78
3.4.2.6 DMA Errors 3-78
3.4.3 Return Read Data Transfer. 3-79
3.4.3.1 Return Read Data to Transaction Buffer . . . 3-79
3.4.3.2 NMI Read Transaction Retries (MEMORY BUSY). 3-83
3.4.3.3 Return Read Data to NBIB 3-83
3.4.3.4 Return Read Data to BCI Data Buffer and BIIC (VAXBI READ DATA CYCLE). 3-83
3.4.3.5 End of VAXBI Transaction 3-84
3.4.3.6 DMA Errors (VAXBI Transaction Retries) . . . 3-85
3.4.4 Parity Generation and Checking 3-86

3.4.4.1 Command/Address and Write Data/Mask Parity 3-86
3.4.4.2 Return Read Data/Status Parity 3-88
3.4.5 Timeouts. 3-90
3.4.6 Read Sequence Faults 3-90
3.5 INTERRUPT (INTR AND IPINTR) OPERATIONS 3-91
3.5.1 Decoding Interrupt Requests. 3-92
3.6 MISCELLANEOUS OPERATIONS 3-96
3.6.1 BIIC Register Read/Write Operations (by Other VAXBI Nodes) 3-96
3.6.2 VAXBI Stop Transactions. 3-96
3.6.3 VAXBI INVAL and BROADCAST Transactions 3-96
3.7 DIAGNOSTIC DATA TRANSFERS. 3-97
3.7.1 BIIC Loopback Requests 3-97
3.7.2 NBIA Wraparound. 3-98
3.7.3 CPU Read/Write to Memory (Flip Address Bits <29> and <22>). 3-99

FIGURES

No. Title Page

1-1 NBI Configuration. 1-2
1-2 NBI Basic Block Diagram 1-4
1-3 DC022 Transaction Buffer Organization 1-6
1-4 CPU Read/Write Data Transfer 1-8
1-5 DMA Read/Write Data Transfers. 1-11
1-6 INTR/IPINTR Operation 1-14
1-7 Interrupt Vector Format 1-15
1-8 SCB Format (Example) 1-16
1-9 Control/Status Register 0 (CSR0) 1-18
1-10 Control/Status Register 0 (CSR1) 1-22
1-11 Vector Registers (BR4VR through BR7VR) 1-24
1-12 Device Register (DTYPE). 1-26
1-13 VAXBI Control/Status Register (BICSR). 1-27
1-14 Bus Error Register (BER) 1-33
1-15 Error Interrupt Control Register (EINTRCSR). . . 1-34
1-16 INTR Destination Register (INTRDES). 1-36
1-17 IPINTR Mask Register (IPINTRMSK) 1-37
1-18 IPINTR/STOP Destination Register (FIPSDES) . . . 1-38
1-19 IPINTR Source Register (IPINTRSRC) 1-39
1-20 Starting Address Register (SADR) 1-40
1-21 Ending Address Register (SADR) 1-41
1-22 VAXBI Control/Status Register (BICSR). 1-42
1-23 Write Status Register (WSTAT). 1-45
1-24 Force IPINTR/STOP Command Register (FIPSCMD) . . 1-46
1-25 User Interrupt Control Register (UINTRCSR) . . . 1-47
1-26 General-Purpose Registers (GPR <3:0>). 1-49

2-1 NBIA and NBIB Input/Output Signals 2-2
2-2 Basic NMI Timing 2-4
2-3 NMI Address Space 2-12

2-4 NMI Write Transaction 2-13
2-5 NMI Read Transaction 2-14
2-6 Basic NMI Arbitration Line Timing 2-16
2-7 NMI Arbitration Line Timing (Typical) 2-18
2-8 MEMORY BUSY Timing 2-19
2-9 Fault Signal Timing 2-22
2-10 Basic VAXBI Timing 2-34
2-11 VAXBI Address Space. 2-37
2-12 VAXBI Node Register Space. 2-38
2-13 VAXBI-Required Registers 2-38
2-14 BIIC-Specific Device Registers 2-39
2-15 VAXBI Write Transaction (Octaword Length). 2-42
2-16 VAXBI Read Transaction (Octaword Length) 2-43
2-17 VAXBI Interrupt (INTR) Transaction 2-46
2-18 VAXBI Identify (IDENT) Transaction 2-49
2-19 VAXBI Interprocessor Interrupt (IPINTR) Transaction. 2-51
2-20 VAXBI STOP Transaction 2-53
2-21 VAXBI Invalidate (INVAL) Transaction 2-55
2-22 Bus Arbitration Request Lines. 2-56
2-23 Arbitration State Diagram. 2-58
2-24 VAXBI Arbitration (Example). 2-60

3-1 NBIA Detailed Block Diagram. 3-3
3-2 NBIB Detailed Block Diagram 3-7
3-3 NBI Powerup 3-15
3-4 Reset by VAXBI Node 3-17
3-5 UNJAM/Programmed NBI INIT 3-19
3-6 NMI Address Decoding and Translation 3-23
3-7 Local Read/Write Command/Address Cycle 3-27
3-8 Local Write Data Cycle 3-29
3-9 Local Read Data Cycle 3-31
3-10 Basic Information Flow Between NMI and VAXBI . . . 3-34
3-11 NMI to VAXBI Command/Address Transfer. 3-37
3-12 NMI to VAXBI Write Data Transfer 3-44
3-13 VAXBI to NMI Return Read Data Transfer 3-50
3-14 Aligned and Unaligned Quadword Read Data Ordering 3-62
3-15 Basic Information Flow Between VAXBI and NMI During DMA Read/Write Operations 3-64
3-16 VAXBI to NMI Command/Address Transfer. 3-67
3-17 VAXBI to NMI Write Data Transfer 3-74
3-18 NMI to VAXBI Return Read Data Transfer 3-80
3-19 INTR/IPINTR Operations 3-93
3-20 FLIP 29/22 Diagnostic Data Transfers 3-101

TABLES

No.	Title	Page
1-1	NBIA Registers	1-17
1-2	Control/Status Register 0 (CSR0) Bit Descriptions	1-19
1-3	Control/Status Register 1 (CSR1) Bit Descriptions	1-22
1-4	NBIB (BIIC) Registers	1-25
1-5	Device Register (DTYPE) Bit Descriptions	1-26
1-6	VAXBI Control/Status Register (BICSR) Bit Descriptions	1-28
1-7	Bus Error Register Bit Descriptions	1-30
1-8	Error Interrupt Control Register (EINTRCSR) Bit Descriptions	1-35
1-9	INTR Destination Register (INTRDES) Bit Descriptions	1-36
1-10	IPINTR Mask Register (IPINTRMSK) Bit Descriptions	1-37
1-11	IPINTR/STOP Destination Register (FIPSDES)	1-38
1-12	IPINTR Source Register (IPINTRSRC) Bit Descriptions	1-39
1-13	Starting Address Register (SADR) Bit Descriptions	1-40
1-14	Ending Address Register (EADR) Bit Descriptions	1-41
1-15	BCI Control/Status Register (BCICSR) Bit Descriptions	1-42
1-16	Write Status Register (WSTAT) Bit Descriptions	1-45
1-17	Force IPINTR/STOP Command Register (FIPSCMD)	1-46
1-18	User Interrupt Control Register (UINTRCSR) Bit Descriptions	1-48
2-1	NMI Signals Connecting to NBIA	2-5
2-2	Data Bus Signals	2-25
2-3	VAXBI Signals	2-30
3-1	BCI Signals	3-9
3-2	NBI Initialization	3-13
3-3	CPU Read/Write Summary	3-21
3-4	DMA Read/Write Summary	3-61

SECTION 1 INTRODUCTION AND SYSTEM OVERVIEW

CHAPTER 1 INTRODUCTION

1.1 MANUAL SCOPE 1-1
1.2 MANUAL ORGANIZATION. 1-1
1.3 RELATED DOCUMENTATION. 1-2
1.4 SYSTEM DESCRIPTION 1-3
1.5 PHYSICAL DESCRIPTION 1-3
1.6 FUNCTIONAL DESCRIPTION 1-7
1.6.1 Console Subsystem. 1-8
1.6.2 Central Processing Unit. 1-10
1.6.2.1 Instruction Box. 1-13
1.6.2.2 Execution Box. 1-16
1.6.2.3 Cache Box. 1-20
1.6.3 Clock Module 1-21
1.6.4 Memory (MBox). 1-23
1.6.4.1 Memory Control Logic 1-24
1.6.4.2 MAR4 Memory Array. 1-25
1.6.5 System Buses 1-25
1.6.5.1 VAX 8800 Memory Interconnect (NMI) 1-25
1.6.5.2 VAX Bus Interconnect (VAXBI) 1-27
1.6.5.3 Visibility Bus (VBus). 1-29
1.6.6 VAX Bus Interconnect and I/O Adapters. 1-30
1.6.7 Power System Complex 1-33
1.6.7.1 876 Power Controller 1-34
1.6.7.2 NBox Port Conditioner. 1-34
1.6.7.3 Module Power Supplies. 1-35
1.6.7.4 Environmental Monitoring Module. 1-35
1.6.7.5 Battery Backup Unit. 1-35

CHAPTER 2 SYSTEM CONTROL

2.1 GENERAL. 2-1
2.2 SYSTEM CONTROL 2-1
2.2.1 Software 2-1
2.2.2 Hardware 2-1
2.3 CONSOLE SOFTWARE COMPONENTS. 2-4
2.3.1 Control Program. 2-4
2.3.1.1 Special Control Program Features 2-4
2.3.1.2 Multiple Command Streams 2-5
2.3.2 File Transfer Program. 2-6
2.3.3 Logical Block Server Program 2-6
2.3.4 Real-Time Interface Driver 2-6
2.4 CONSOLE SUPPORT MICROCODE (CSM). 2-6
2.4.1 Console Support Microcode Structure. 2-6
2.4.2 CSM Data Transfers/Protocol. 2-7
2.4.3 Console Support Microcode Entry Points 2-7

2.5 OPERATIONAL MODES. 2-8
2.5.1 Console Mode 2-8
2.5.2 Program Mode 2-9
2.6 OPERATOR/CONSOLE INTERACTION 2-9
2.6.1 Console State Bits 2-9
2.6.1.1 Command Validity 2-10
2.6.1.2 Saving Console State 2-10
2.6.2 Console Commands 2-10
2.6.2.1 Executing Console Commands 2-12
2.6.3 Console/Operator Display 2-13
2.6.3.1 Local Display During Remote Operation. . . . 2-13
2.6.3.2 Console Command Language Display Prompts . . 2-15
2.6.4 System Logfile 2-15
2.6.4.1 Displaying the Logfile 2-15
2.6.4.2 Logical Terminals/Logfile Integrity. 2-15
2.6.4.3 Saving the Logfile 2-16
2.7 POWER-UP/DOWN SEQUENCING 2-16
2.7.1 Power On 2-16
2.7.2 Power Fail 2-17
2.7.3 Powerdown. 2-17
2.7.4 Warm Restart 2-17

CHAPTER 3 SYSTEM OPERATION

3.1 INTRODUCTION 3-1
3.2 MICROCODE. 3-1
3.2.1 Microcode Characteristics. 3-1
3.2.1.1 Functionality. 3-1
3.2.1.2 Operation. 3-1
3.2.1.3 Structure. 3-1
3.2.2 Microcode Control. 3-2
3.2.2.1 Interrupt and Processor Register (INPR). . . . 3-2
3.2.2.2 Condition Code and Branch (CCBR) 3-3
3.2.2.3 Gateway Control (GWYC) 3-3
3.2.2.4 Microtrap (UTRP) 3-3
3.2.2.5 Microbranch Slices (UBRS). 3-3
3.2.3 Pipelining 3-3
3.2.4 Microtraps 3-4
3.2.5 Micromatch 3-5
3.2.5.1 Stop on Match. 3-5
3.2.5.2 Trap on Match. 3-6
3.3 MEMORY ADDRESSING AND READ/WRITE OPERATIONS. . . . 3-6
3.3.1 Virtual Addresses. 3-6
3.3.1.1 Layout 3-6
3.3.1.2 Format 3-6
3.3.2 Physical Addresses 3-9
3.3.3 Address Translation. 3-10
3.3.3.1 Page Table Entry 3-10
3.3.4 Cache Operation. 3-15
3.3.4.1 Translation Buffer 3-15
3.3.4.2 Cache. 3-16
3.3.4.3 NMI Interface. 3-17

3.3.5 Read/Write Operations. 3-17
3.3.6 Device Address Selection 3-19
3.3.6.1 Transaction Significant Address Bits 3-19
3.4 INTERRUPTS AND EXCEPTIONS. 3-20
3.4.1 Servicing. 3-20
3.4.1.1 NMI Interrupt Enable Register. 3-20
3.4.1.2 Types of Interrupts. 3-20
3.4.1.3 Servicing the Interrupt. 3-20
3.4.2 Priority Levels. 3-20
3.4.3 System Control Block (SCB) Format. 3-21
3.4.3.1 SCB Pagination 3-23
3.4.3.2 Offsetable Devices 3-23
3.4.3.3 VAXBI Node Direct Connected Devices. 3-23
3.4.3.4 SCB Format 3-24
3.4.4 Machine Check Exception. 3-25
3.4.4.1 Types of Exceptions. 3-25

CHAPTER 4 DIAGNOSTIC AND MAINTENANCE AIDS

4.1 INTRODUCTION 4-1
4.2 GENERAL. 4-1
4.3 DIAGNOSTICS. 4-3
4.3.1 Console Selftest 4-3
4.3.2 Microdiagnostics 4-3
4.3.2.1 CSM Commands 4-4
4.3.2.2 Status and Error Information 4-6
4.3.2.3 Micromonitor Error Messages. 4-8
4.3.3 Macrodiagnostics 4-8
4.3.4 Customer Runnable Diagnostics. 4-9
4.3.4.1 Auto-Test Mode 4-10
4.3.4.2 Menu Mode. 4-10
4.3.5 Remote Diagnostics 4-10
4.4 POWER/ENVIRONMENTAL SYSTEM 4-10
4.4.1 Module Placement Verification. 4-10
4.4.1.1 Module Key Test. 4-10
4.4.1.2 Module Placement 4-12
4.4.2 Power Monitoring/Error Reporting 4-12
4.4.2.1 Default Mode Error Reporting 4-13
4.4.2.2 Operational Error Reporting. 4-13
4.4.3 Voltage Margining. 4-14
4.5 MAINTENANCE AIDS 4-16
4.5.1 Machine Check Logout Stack 4-16

FIGURES

No.	Title	Page
1-1	VAX 8800 System Major Component Locations (Rear View)	1-4
1-2	Cardcage Module Layout (Front View)	1-6
1-3	Simplified Block Diagram of the VAX 8800 System	1-8
1-4	Simplified Block Diagram of the Console Subsystem	1-10
1-5	Simplified Block Diagram of Single CPU	1-12
1-6	Simplified Block Diagram of the CPU IBox	1-13
1-7	Simplified Block Diagram of the CPU Execution Box	1-17
1-8	Simplified Block Diagram of the CBox	1-21
1-9	Simplified Block Diagram of the Clock Module	1-22
1-10	Simplified Block Diagram of the MBox	1-24
1-11	Simplified Block Diagram of the VAX 8800 Memory Interconnect	1-26
1-12	Simplified Diagram of VAX Bus Interconnect (Maximum Configuration)	1-28
1-13	I/O Interconnect and Adapters	1-31
1-14	NMI-to-VAXBI Adapter	1-32
1-15	Simplified Block Diagram of the Power System Complex	1-34
2-1	VAX 8800 System Hardware and Software Control Components	2-2
2-2	Console Operational Modes	2-8
2-3	Local/Remote Display Character Flow	2-14
3-1	Simplified Pipelining	3-4
3-2	Virtual Address Space Layout	3-7
3-3	Virtual Address Format	3-8
3-4	Physical Address Space Layout	3-9
3-5	MAP Enable Register Bit Configuration	3-10
3-6	Page Table Entry Bit Configuration	3-10
3-7	System Space Virtual-to-Physical Address Translation	3-12
3-8	Process Space (P0 Region) Virtual-to-Physical Address Translation	3-13
3-9	Process Space (P1 Region) Virtual-to-Physical Address Translation	3-14
3-10	CBox Functional Components	3-15
3-11	NMI Address Selection	3-18
3-12	NMI Address Bit Significance	3-19
3-13	NBI I/O Adapter SCB Vector Offset Format	3-24

4-1 Bottom-Up Testing. 4-2
4-2 Module Keying Test Simplified Block Diagram. . . 4-11
4-3 Module Key Test Connections. 4-12
4-4 Margin Enable and Margin Hi Lo Registers 4-15
4-5 Machine Check Logout Stack 4-17
4-6 CBox Error Register. 4-18
4-7 IBox Error Register. 4-18
4-8 EBox Error Register. 4-19
4-9 NMI Interrupt Control Register 4-19
4-10 NMI Fault Summary. 4-20
4-11 NMI Silo Data. 4-21
4-12 NMI Error Address Register 4-21
4-13 Cache ON Register. 4-22
4-14 Machine Check Status 4-22
4-15 Revision 1/2 Registers 4-22

TABLES

No. Title Page

1-1 Technical Description Manual Organization. 1-2
1-2 Related Documentation. 1-3
1-3 VAX 8800 System Physical Characteristics 1-5
1-4 Cabinet Module Identification. 1-7
1-5 Power Supply Identification. 1-7
1-6 VAX 8800 Processor Functional Units/Data Bus Descriptions 1-11
1-7 System Clocks. 1-23
1-8 MCL Command Operations 1-24
1-9 NMI Function Descriptions. 1-27
1-10 VAXBI Function Descriptions. 1-29
1-11 Optional VAX Bus Interconnect Adapters 1-30
1-12 NBIA Registers 1-33
1-13 NBIB Registers 1-33
1-14 NBox Modules 1-35

2-1 Hardware and Software Component Description. . . . 2-3
2-2 Major Sections of CSM Code 2-7
2-3 Console Support Microcode Entry Points 2-8
2-4 Bit Examples 2-9
2-5 Console Command Overview 2-11
2-6 Console Command Language Prompts 2-15
2-7 Module Power Supply Turn-On Sequence 2-17

3-1 VAX 8800 System Microtraps 3-5
3-2 Page Table Entry Bit Description 3-11
3-4 Translation Buffer Field Description 3-15
3-4 Hardware Interrupt Priority Level Assignments. . 3-21
3-5 System Control Block Page 0 (000--1FF) 3-22
3-6 Machine Check Exception Examples 2-25

4-1 TEMP Register Addresses for Use with Diagnostic CSM 4-5
4-2 Microcode Error Register Addresses 4-6
4-3 Macrodiagnostic Tests. 4-9

EXAMPLES

No. Title Page

4-1 Sample Microdiagnostic Display Output. 4-7
4-2 Sample Microdiagnostic Error Display 4-7
4-3 EMM Warning Message. 4-14

SECTION 2 SYSTEM BUS SUMMARY

CHAPTER 1 MEMORY INTERCONNECT (NMI)

1.1 INTRODUCTION 1-1
1.2 BASIC FUNCTIONS 1-4
1.3 NMI SIGNALS AND TIMING 1-5
1.4 NMI ADDRESS SPACE. 1-18
1.5 READ/WRITE TRANSACTIONS. 1-25
1.6 INTERLOCKED OPERATIONS 1-32
1.7 BUS ARBITRATION. 1-33
1.8 INTERRUPTS 1-39
1.8.1 NMI Interrupt Priority Levels. 1-39
1.8.2 Device Interrupts. 1-40
1.8.3 NMI Faults 1-41
1.9 NMI ERRORS 1-42

CHAPTER 2 VAX BUS INTERCONNECT (VAXBI)

2.1 INTRODUCTION 2-1
2.2 BASIC FUNCTIONS. 2-4
2.3 VAXBI SIGNALS AND TIMING 2-5
2.4 VAXBI ADDRESS SPACE. 2-11
2.4.1 Memory Address Space 2-11
2.4.2 I/O Address Space. 2-11
2.4.3 Address Selection. 2-12
2.5 BASIC VAXBI TRANSACTION FORMAT 2-18
2.5.1 Command/Address Cycle. 2-18
2.5.2 Embedded Arbitration Cycle 2-18
2.5.3 Data Cycles. 2-19
2.5.4 Bus Parity 2-19
2.6 READ/WRITE TRANSACTIONS. 2-20
2.6.1 Write Data Cycles. 2-20
2.6.2 Read Data Cycles 2-20
2.6.3 Nonexistent Addresses. 2-23

2.6.4 Stalls 2-23
2.6.5 Retries. 2-24
2.7 BROADCAST TRANSACTIONS 2-24
2.8 INVALIDATE TRANSACTIONS. 2-26
2.9 INTERRUPT OPERATION (INTR, IDENT, AND IPINTR TRANSACTIONS 2-28
2.9.1 Interrupt (INTR) Transactions. 2-29
2.9.2 Identify (IDENT) Transactions. 2-31
2.9.3 Interprocessor Interrupt (IPINTR) Transaction. 2-34
2.10 STOP TRANSACTIONS. 2-36
2.11 BUS ARBITRATION AND CONTROL. 2-38
2.11.1 Bus Requests 2-38
2.11.2 Arbitration Modes. 2-39
2.11.3 Arbitration Control. 2-39
2.11.4 Extending a Transaction. 2-41
2.11.5 Special Mode Functions 2-41
2.12 VAXBI ERRORS 2-42
2.12.1 Parity Checking. 2-42
2.12.2 Transmit Check Error Detection 2-43
2.12.3 Protocol Checking. 2-44

CHAPTER 3 VISIBILITY BUS (VBUS)

3.1 INTRODUCTION 3-1
3.2 BASIC FUNCTIONS. 3-1
3.3 VBUS SIGNALS 3-1
3.4 VBUS REGISTERS 3-4
3.5 MODULE VBUS CHANNEL CIRCUITRY. 3-7
3.5.1 Minimum Configuration. 3-7
3.5.2 Expanded Configuration 3-7
3.6 VBUS ADDRESS/DATA SUMMARY. 3-10
3.7 VBUS CONSOLE COMMANDS. 3-10

FIGURES

No. Title Page

1-1 Memory Interconnect (NMI). 1-2
1-2 Basic NMI Timing 1-5
1-3 NMI Signals. 1-6
1-4 NMI Address Space. 1-20
1-5 NMI Address Bits 1-23
1-6 NMI Address Selection. 1-24
1-7 NMI Write Transaction. 1-26
1-8 NMI Read Transaction 1-27
1-9 NMI Write Transaction Types. 1-29

1-10 NMI Read Transaction Types 1-30
1-11 Basic NMI Arbitration Line Timing. 1-33
1-12 NMI Arbitrator Operation 1-36
1-13 Detailed NMI Arbitration Line Timing (Typical) . 1-37
1-14 MEMORY BUSY Timing 1-38
1-15 Fault Signal Timing. 1-42

2-1 VAX Bus Interconnect (VAXBI) 2-2
2-2 VAXBI Signals. 2-6
2-3 Basic VAXBI Timing 2-10
2-4 VAXBI Address Space. 2-13
2-5 VAXBI Node Register Space. 2-14
2-6 VAXBI Required Registers 2-15
2-7 BIIC-Specific Device Registers 2-16
2-8 VAXBI Read/Write Address Bits. 2-17
2-9 Basic VAXBI Transaction Format 2-18
2-10 VAXBI Write Transaction (Octaword Length). . . . 2-21
2-11 VAXBI Read Transaction (Octaword Length) 2-22
2-12 VAXBI Broadcast (BDCST) Transaction (Octaword Length). 2-25
2-13 VAXBI Invalidate (INVAL) Transaction 2-27
2-14 VAXBI Interrupt (INTR) Transaction 2-30
2-15 VAXBI Identify (IDENT) Transaction 2-33
2-16 VAXBI Interprocessor Interrupt (IPINTR) Transaction. 2-35
2-17 VAXBI STOP Transaction 2-37
2-18 Bus Arbitration Request Lines. 2-38
2-19 Arbitration State Diagram. 2-40
2-20 VAXBI Arbitration (Example). 2-42

3-1 Visibility Bus (VBus) and VBus Control (on CLK Module). 3-2
3-2 VBus Control Register. 3-5
3-3 VBus Access Register 3-6
3-4 VBus Channel in CPU Module (Minimum Configuration) 3-8
3-5 VBus Channel in CPU Module (Expanded Configuration) 3-9

TABLES

No. Title Page

1-1 Glossary of NMI Terms. 1-3
1-2 NMI Signal Descriptions. 1-10
1-3 I/O Registers in NBI and Memory Controller . . . 1-21
1-4 NMI Interrupt Priority Levels (IPLs) 1-39
1-5 NMI Errors 1-43

2-1 Glossary of VAXBI Terms. 2-3
2-2 VAXBI Signal Descriptions. 2-7

3-1 VBUS Signal Descriptions 3-3
3-2 VBus Control Register Bit Descriptions 3-5
3-3 VBus Directory (Excerpt) 3-11

SECTION 3 CONSOLE SUBSYSTEM

CHAPTER 1 INTRODUCTION

1.1 GENERAL. 1-1
1.2 RELATED DOCUMENTATION AND REFERENCES 1-1
1.3 FUNCTION AND PURPOSE 1-2
1.4 SUBSYSTEM COMPONENTS 1-2
1.5 CONSOLE/VAX 8800 INTERACTION 1-4
1.5.1 Power-Up Mode. 1-4
1.5.2 Console I/O. 1-4
1.5.3 VAX 8800 State Description 1-6
1.5.3.1 Power Off. 1-6
1.5.3.2 Clock Stopped/WCS Invalid. 1-6
1.5.3.3 Clock Running/WCS Invalid. 1-6
1.5.3.4 Clock Running/WCS Valid. 1-6
1.5.4 Console State Description. 1-7
1.6 CONSOLE SOFTWARE COMPONENTS. 1-9
1.6.1 Control Program. 1-9
1.6.2 Logical Block Server Program 1-9
1.6.3 Real-Time Interface Driver 1-9
1.7 CONSOLE/VAX 8800 POWER SEQUENCE. 1-10
1.7.1 Powerup (Refer to Figure 1-3). 1-10
1.7.2 EMM/Console Initialize (Refer to Figure 1-4). 1-12
1.7.3 Restart/Boot/Halt (Refer to Figure 1-5). 1-17
1.7.4 Power Fail (Refer to Figure 1-6) 1-19
1.7.5 Powerdown (Refer to Figure 1-7). 1-21

CHAPTER 2 FUNCTION DESCRIPTION

2.1 GENERAL. 2-1
2.2 REAL-TIME INTERFACE (RTI). 2-1
2.2.1 Programmable Peripheral Interface (PPI). 2-3
2.2.1.1 Port A 2-3
2.2.1.2 Port B 2-3
2.2.1.3 Port C 2-6
2.2.1.4 PPI Control. 2-8
2.2.2 Serial Line Port 2-10
2.2.2.1 ECPI Registers 2-10
2.2.2.2 Data Transfer and Status Registers 2-14

2.3 CONSOLE INTERFACE. 2-16
2.3.1 Buffer Translate and Synchronize 2-16
2.3.2 Console Address Decode 2-16
2.3.3 Console Sequencer (CSEQ MCA) 2-17
2.3.4 Terminal Register/Interval Clock (TRIC) MCA. . 2-17
2.3.4.1 Program Mode 2-19
2.3.4.2 Console Mode 2-20
2.3.5 Data Output Mux. 2-20
2.3.6 Control Registers. 2-21
2.3.7 Visibility Bus Control 2-22
2.3.8 Console Interrupt Generation 2-23
2.3.9 Power Status 2-23
2.4 CONSOLE/VAX 8800 INTERACTION 2-24
2.4.1 Initialization 2-24
2.4.1.1 Turn ON and Monitor System Power/Reset EMM. 2-24
2.4.1.2 Console Poweron. 2-27
2.4.1.3 Load and Run Console Power-Up Software . . . 2-28
2.4.1.4 Sequenced Power Application. 2-32
2.4.1.5 Initialize Hardware. 2-36
2.4.1.6 Test and Checkout. 2-44
2.4.1.7 Load RAMs and DRAMs. 2-46
2.4.2 VAX 8800 CPU Control 2-63
2.4.2.1 Console Sequencer. 2-64
2.4.2.2 Control Registers. 2-68
2.4.3 Data Transfers 2-70
2.5 CONSOLE/VAX 8800 CLOCKS AND TIMING 2-73
2.5.1 One-MHz Clock. 2-73
2.5.2 Interval Clock 2-75
2.5.3 CPU Timeouts 2-78
2.5.4 Visibility Bus 2-79

CHAPTER 3 DETAILED DESCRIPTION

3.1 GENERAL. 3-1
3.2 TERMINAL REGISTER INTERVAL CLOCK (TRIC). 3-1
3.3 CONSOLE SEQUENCER MCA (CSEQ) 3-9
3.3.1 Console Strobe Sequencer 3-9
3.3.2 Read Acknowledge 3-9
3.3.3 Console Write Sequencer. 3-9
3.3.4 Control Store Load Sequencer 3-9
3.4 CONSOLE/VAX 8800 REGISTER SUMMARY. 3-17
3.4.1 Console Registers (Refer to Figure 3-7). 3-17
3.4.2 VAX 8800 CPU Registers (Refer to Figure 3-8) . . 3-24
3.5 CONSOLE CABLING. 3-28

FIGURES

No.	Title	Page
1-1	Simplified Block Diagram of the Console Subsystem	1-3
1-2	Modes of Operation	1-5
1-3	Power-Up Sequence	1-11
1-4	EMM/Console Initialize	1-13
1-5	Restart/Boot/Halt	1-18
1-6	Power-Fail Sequence	1-20
1-7	Powerdown Procedure	1-22
2-1	Console Subsystem Functional Block Diagram	2-2
2-2	PPI Port A Format	2-3
2-3	PPI Port B Format	2-3
2-4	PPI Port C Format	2-6
2-5	PPI Control Register Format	2-9
2-6	ECPI Mode 1 Register	2-10
2-7	ECPI Mode 2 Register	2-12
2-8	ECPI Command Register	2-13
2-9	Serial Line Port Data and Status Registers	2-14
2-10	RXDB, TXDB, and DBCS	2-18
2-11	Control Registers	2-21
2-12	VBus Control and Access Registers	2-22
2-13	System Power-On Sequence	2-25
2-14	Environmental Monitoring Module Reset Sequence	2-26
2-15	Console Power-On Events	2-27
2-16	Serial Line Port Data Transfer Registers	2-28
2-17	Load/Run Console Power-Up Software Events	2-30
2-18	Sequenced Power Application Events	2-32
2-19	Console Interconnect Loopback Testing Through Ports A, B, and C	2-37
2-20	Console Interconnect Loopback Testing Through Ports B and C	2-37
2-21	Interface Data Path Loopback Test of Unbuffered Data	2-38
2-22	Interface Data Path Loopback Test of Buffered Data Through RXDB and TXDB	2-39
2-23	Console Sequencer Enable Logic	2-40
2-24	Control Register Initialization	2-41
2-25	Hardware Initialization Events	2-42
2-26	Test and Checkout Events	2-44
2-27	VBus Parity Bits	2-49
2-28	DRAM Address	2-51
2-29	MNI Control Store Address	2-53
2-30	RAM Loading Simplified Block Diagram	2-56
2-31	RAM/DRAM Loading Events	2-57
2-32	Console/Interface Timing Signals	2-64
2-33	Write Sequence	2-65
2-34	Read Sequence (Setup)	2-66
2-35	Read Sequence (Data Out)	2-67

2-36 Simplified Diagram of Control Register 0 2-68
2-37 Simplified Diagram of Control Register 1 2-69
2-38 Simplified Diagram of Control Register 2 2-69
2-39 Data Transfer Control Interface-to-VAX 8800 CPU. 2-71
2-40 Data Transfer VAX 8800 CPU-to-Console Interface. 2-72
2-41 1 MHz Clock Generation 2-74
2-42 Interval Clock Registers Bit Configuration . . . 2-75
2-43 Interval Clock Simplified Block Diagram. 2-77
2-44 Simplified CPU Timeout 2-78
2-45 Clock Status and Timeout Register (CST). 2-78
2-46 VBus Control/Data Signals. 2-79
2-47 VBus Logic Simplified Block Diagram. 2-80
3-1 TRIC MCA Block Diagram 3-2
3-2 TRIC MCA Pin Layout. 3-3
3-3 TRIC MCA Body Drawing. 3-4
3-4 CSEQ MCA Block Diagram 3-10
3-5 CSEQ MCA Pin Layout. 3-11
3-6 CSEQ MCA Body Drawing. 3-12
3-7 Console Registers. 3-17
3-8 VAX 8800 CPU Registers 3-24
3-9 Console Subsystem Cabling Diagram. 3-29

TABLES

No. Title Page

2-1 PPI Port B Bit Description 2-4
2-2 PPI Port C bit Description 2-6
2-3 PPI Control Register Bit Description 2-9
2-4 ECPI Mode 1 Register Bit Description 2-11
2-5 ECPI Mode 2 Register Bit Description 2-12
2-6 ECPI Command Register Bit Description. 2-13
2-7 Serial Line Port Data and Status Registers Bit Description. 2-15
2-8 Serial Line Port Data Transfer Registers Bit Description. 2-29
2-9 Key Initialization Signal/Functions. 2-62
2-10 ICCS Bit Configuration 2-76

3-1 TRIC MCA Pin Assignments 3-5
3-2 TRIC MCA Signal Descriptions 3-6
3-3 CSEQ MCA Pin Assignments 3-13
3-4 CSEQ MCA Signal Descriptions 3-14
3-5 Console Cable List 3-28

SECTION 4 POWER SYSTEM COMPLEX

CHAPTER 1 GENERAL DESCRIPTION

1.1 INTRODUCTION 1-1
1.2 SYSTEM COMPONENTS. 1-4
1.2.1 876A Power Controller. 1-5
1.2.2 NBox Power Converter 1-5
1.2.3 Modular Power System (MPS) 1-6
1.2.4 Environmental Monitoring Module. 1-6
1.2.5 Cooling System 1-7
1.2.6 Battery Backup Unit H7231-M. 1-7
1.3 MECHANICAL CONFIGURATION 1-8
1.3.1 876A Power Controller. 1-11
1.3.2 NBox Port Conditioner. 1-11
1.3.3 MPS Modules (Regulators) and Cage. 1-11
1.3.4 Battery Backup Unit. 1-12
1.3.5 Air Flow System. 1-12
1.4 POWER DISTRIBUTION 1-13
1.4.1 AC Power 1-13
1.4.2 DC Power 1-15
1.4.3 Controls and Breakers. 1-17
1.4.3.1 Controls 1-17
1.4.3.2 Circuit Breakers 1-17
1.5 AC POWER SPECIFICATIONS. 1-19
1.5.1 Electrical Requirements. 1-19
1.5.1.1 AC Power Sources 1-19
1.6 FAULT AND STATUS INDICATORS. 1-21
1.6.1 876A Power Controller. 1-21
1.6.2 NBox 1-21
1.6.2.1 H7170 Built-In Test Equipment. 1-23
1.6.2.2 ILM Built-In Test Equipment. 1-23
1.6.3 Modular Power Supply Regulators. 1-24
1.6.4 Environmental Monitoring Module. 1-25
1.6.5 System Console Device. 1-26

CHAPTER 2 FUNCTIONAL DESCRIPTION

2.1 INTRODUCTION 2-1
2.2 POWER SYSTEM BLOCK DIAGRAM 2-1
2.3 SIMPLIFIED OPERATION 2-3
2.3.1 876A Power Controller. 2-3
2.3.2 NBox 2-4
2.3.2.1 H7170 Power Converter. 2-4
2.3.2.2 Control Start-up Power Module (CSP). 2-5
2.3.2.3 Interface Logic Module (ILM) 2-5
2.3.2.4 New Box Translator Module (NBT). 2-5
2.3.3 Modular Power System (MPS) 2-6
2.3.4 Battery Backup Unit (BBU Model H7231-M). 2-8
2.3.4.1 BBU Control. 2-9
2.3.5 Environmental Monitoring Module (EMM). 2-11

2.3.6 Power System Monitoring. 2-13
2.3.6.1 Key Monitoring 2-14
2.3.6.2 Regulator Control. 2-14
2.3.6.3 BBU Control. 2-14
2.3.6.4 Air Flow Status. 2-14
2.3.6.5 AC/DC LO Signals 2-15
2.3.6.6 Regulator Overtemp. and CPU Cabinet Temperature (Thermistor Volts) 2-15
2.4 POWER SEQUENCES. 2-16
2.4.1 Circuit Breakers 2-16
2.4.2 Summary of Power-Up Sequence 2-18
2.5 SYSTEM POWER-UP FLOWCHART (FIGURE 2-5) 2-24
2.6 CONSOLE POWER-DOWN FLOWCHART (FIGURE 2-6). . . . 2-29
2.7 POWER-DOWN/POWER INTERRUPT WITH BBU FLOWCHART (FIGURE 2-7) 2-31
2.8 POWERUP FROM BBU FLOWCHART (FIGURE 2-8). 2-35

CHAPTER 3 DETAILED DESCRIPTION

3.1 INTRODUCTION 3-1
3.2 BLOCK DIAGRAM OF THE VAX 8800 POWER SYSTEM 3-1
3.3 876A POWER CONTROLLER. 3-3
3.4 NBOX POWER CONVERTER ASSEMBLY. 3-6
3.4.1 NBox Modules 3-8
3.4.1.1 H7170. 3-8
3.4.2 CSP. 3-9
3.4.2.1 ILM. 3-9
3.4.2.2 NBT Module 3-14
3.4.3 Modular Power System -- MPS. 3-16
3.4.3.1 H7186 +5.0-Volt Regulator. 3-18
3.4.3.2 Side Panel 3-20
3.4.3.3 H7186 Main Board 3-21
3.4.4 H7187 -2.0-Volt Regulator. 3-22
3.4.5 H7180 -5.2-Volt Regulator. 3-22
3.4.5.1 H7180 Side Panel 3-24
3.4.5.2 H7180 Main PC Board. 3-24
3.4.6 H7189 BIP Regulator. 3-26
3.4.6.1 H7189 Functional Description 3-29
3.4.7 MPS Regulator BITE Indicators. 3-33
3.4.8 Buses and Backplanes 3-35
3.4.9 MPS Backplane. 3-35
3.4.9.1 300-Vdc Buses. 3-40
3.4.10 Environmental Monitoring Module. 3-41
3.4.10.1 8085A Microprocessor System. 3-43
3.4.10.2 Electric Key Monitor 3-44
3.4.10.3 Regulator Control Circuits 3-45
3.4.10.4 Regulator On/Off Control Circuits. 3-46
3.4.10.5 Regulator Margin Control Circuits. 3-46
3.4.11 Status Registers 3-48
3.4.11.1 AC/DC LO Circuits. 3-49
3.4.11.2 Total-Off Control and Indicator Circuits . . 3-52

3.4.11.3 Temperature Sensing Circuits 3-53
3.4.11.4 Voltage Measuring Circuit. 3-55
3.4.11.5 EMM/Console Voltage Tests. 3-58
3.4.11.6 Battery Backup Unit (BBU) Control. 3-58
3.4.12 Battery Backup Unit (H7231-M). 3-59
3.4.13 Air Flow Sensing Circuit 3-63
3.5 COOLING SUBSYSTEM. 3-65

FIGURES

No. Title Page

1-1 VAX 8800 System Physical Layout (Front View) . . . 1-2
1-2 VAX 8800 Power System Block Diagram (60 Hz). . . . 1-3
1-3 VAX 8800 Power System Block Diagram (50 Hz). . . . 1-4
1-4 VAX 8800 CPU Cabinet - Front View. 1-9
1-5 VAX 8800 CPU Cabinet - Rear View 1-10
1-6 876A Rear View Showing Receptacles 1-14
1-7 DC Power Section Block Diagram 1-16
1-8 VAX 8800 Power System Circuit Location Diagram. 1-18
1-9 876A Power Controller Front Panel. 1-21
1-10 NBox Front-Panel Indicators. 1-22
1-11 Indicators for the Modular Power Regulators. . . 1-24
1-12 EMM Front-Panel Indicators 1-25

2-1 VAX 8800 Power System Block Diagram. 2-2
2-2 MPS Backplane Configuration (Rear View). 2-7
2-3 Battery Backup Subsystem Functional Block Diagram. 2-10
2-4 EMM Functional Block Diagram 2-12
2-5 System Power-Up Flowchart. 2-20
2-6 System Power-Down Flowchart. 2-28
2-7 Powerdown/Power Interrupt with BBU Flowchart . . 2-30
2-8 AC Powerup from BBU Operation Flowchart. 2-34

3-1 Power System Block Diagram 3-2
3-2 876A Front and Rear Panels 3-4
3-3 876A Power Controller Block Diagram. 3-5
3-4 NBox System Interconnect 3-7
3-5 ILM PC Board Signals 3-10
3-6 NBT PC Board Signals 3-15
3-7 MPS Regulator Configuration. 3-17
3-8 H7186 Block Diagram. 3-19
3-9 H7180 Block Diagram. 3-23
3-10 H7189 Block Diagram. 3-27
3-11 BITE Indicators. 3-34
3-12 Organization of the Power System 3-37
3-13 MPS I Backplane. 3-38
3-14 MPS II Backplane 3-39
3-15 EMM Block Diagram. 3-42
3-16 Voltage Margining Circuit. 3-47

3-17 AC/DC LO Timing Diagram. 3-50
3-18 AC/DC LO Circuit 3-51
3-19 Temperature Sensing Circuit. 3-54
3-20 Voltage Measuring Circuit. 3-56
3-21 Voltage Measuring Technique. 3-57
3-22 BBU Block Diagram. 3-62
3-23 Air Flow Sensing Circuit 3-64
3-24 Air Flow Path. 3-66

TABLES

No. Title Page

1-1 Power System Components. 1-4
1-2 NBox Modules 1-5
1-3 876A Power Distribution. 1-15
1-4 VAX 8800 Circuit Breakers. 1-17
1-5 H7170 Status Indicators. 1-23
1-6 MPS Regulator Indicators 1-24
1-7 EMM Magnetic Status Indicator Codes. 1-26

2-1 876A Power Distribution. 2-4
2-2 Modules Using CSP Bias Voltages. 2-5
2-3 Voltage Regulators 2-7
2-4 System Circuit Breakers. 2-17

3-1 876A AC Power Distribution 3-3
3-2 NBox Modules 3-8
3-3 CSP Voltages 3-9
3-4 VAX 8800 MPS Regulators. 3-18
3-5 H7186 Side-Panel Components and Interconnects. . . 3-20
3-6 H7186 Main Board Circuits and Interconnects. . . . 3-21
3-7 H7180 Side Panel Components and Interconnects. . . 3-24
3-8 H7180 Main PCB Circuits and Interconnects. 3-24
3-9 H7189 Module Functions 3-28
3-10 H7189 Outputs. 3-31
3-11 H7189 Module I Circuits and Interconnects. 3-32
3-12 H7189 Module II Circuits and Interconnects 3-32
3-13 MPS Regulator Connectors 3-36
3-14 300-V Buses, Power Sources, and Loads. 3-40
3-15 Battery Backup Interface Signals 3-61

SECTION 5 CLOCK MODULE

CHAPTER 1 INTRODUCTION

1.1 BASIC OPERATION. 1-1
1.2 BASIC COMPONENTS AND TIMING. 1-3
1.3 CLOCK CONTROL (BY CONSOLE) 1-5
1.4 CLOCK STALLS 1-7
1.5 CLOCK STATUS 1-7

CHAPTER 2 FUNCTIONAL DESCRIPTION

2.1 DETAILED BLOCK DIAGRAM 2-1
2.1.1 Oscillator 2-1
2.1.2 Phase Generator. 2-2
2.1.3 Clock Control Logic. 2-2
2.1.4 Clock Distribution Circuits. 2-2
2.2 CLOCK GENERATOR INITIALIZATION 2-7
2.3 SYSTEM CLOCK PERIOD CONTROL. 2-8
2.3.1 Phase-Locked Loop Operation. 2-8
2.3.2 Changing Clock Period. 2-11
2.4 SYSTEM CLOCK START/STOP/BURST CONTROL. 2-12
2.4.1 Starting the Clocks. 2-13
2.4.2 Stopping the Clocks Unconditionally. 2-13
2.4.3 Stopping the Clocks on Micromatch/Scope Sync Generation 2-16
2.4.4 Bursting the Clocks. 2-17
2.4.5 Single-Stepping the Clocks 2-17
2.4.6 Single-Stepping the B CLK. 2-18
2.5 SLOW CLOCK GENERATION AND CONTROL. 2-19
2.6 CLOCK CONSOLE COMMANDS 2-20

FIGURES

No. Title Page

1-1 Clock Generator (and Console Interface) on Clock Module. 1-2
1-2 System Clock Timing Diagram. 1-4
1-3 Clock Control Register 1-8
1-4 Burst Count Register 1-9
1-5 Clock Period Register. 1-9
1-6 Clock/Timeout Status Register. 1-10

2-1 Clock Generator (Detailed Block Diagram) 2-3
2-2 Simplified Clock Frequency Control Circuitry 2-9
2-3 Simplified Clock Start/Stop/Burst Control Logic. 2-14
2-4 Start/Stop/Burst Control Timing Diagram. 2-15
2-5 Micromatch and Scope Sync Timing Diagram 2-16
2-6 Single-Stepping B CLK Timing Diagram 2-18
2-7 Slow Clock Timing Diagram. 2-19

TABLES

No.	Title	Page
1-1	System Clocks	1-1
1-2	Clock Control Register Bit Descriptions	1-8
1-3	Clock/Timeout Status Register Bit Descriptions	1-10
2-1	Clock Generator Inputs	2-4
2-2	Clock Generator Outputs	2-5
2-3	Clock Console Commands	2-20

SECTION 6 INSTRUCTION BOX (IBOX)

CHAPTER 1 INTRODUCTION

1.1 OVERVIEW 1-1
1.1.1 Dual-Processor Configuration 1-1
1.2 LOGIC ELEMENTS 1-2
1.2.1 Physical Implementation. 1-2
1.2.2 Instruction Buffer (IB). 1-5
1.2.2.1 Writing the IB 1-5
1.2.2.2 Reading the IB 1-5
1.2.3 IB Manager 1-6
1.2.3.1 IB Read/Write Control. 1-6
1.2.3.2 Computing Amount of IB Data Consumed 1-6
1.2.4 Decoder Logic. 1-7
1.2.4.1 Decoder RAMs 1-7
1.2.4.2 Special Address Encoder. 1-8
1.2.5 Microsequencer Logic 1-8
1.2.6 Control Store. 1-9
1.2.7 Condition Code and Macrobranch Logic 1-9
1.2.7.1 PSL CC Bits. 1-9
1.2.7.2 CPU State Flags. 1-10
1.2.8 Interrupt and Processor Register Logic 1-10
1.2.8.1 Interrupt Logic. 1-10
1.2.8.2 Processor Register Logic 1-10
1.2.9 File Address Generator 1-10
1.2.10 Gateway Control Logic. 1-11
1.2.10.1 Primary Functions. 1-11
1.3 IBOX BUSES 1-12
1.3.1 Cache Data Bus 1-12
1.3.2 IB Data Bus. 1-12
1.3.3 Cons Bidi Data Bus 1-12
1.4 IBOX RESIDENT INTERNAL PRIVILEGED REGISTERS (IPRs) 1-13
1.4.1 VAX Architecture IPRs. 1-13
1.4.2 VAX 8800-Specific IPRs 1-14
1.4.2.1 NMI Interrupt Control Register (NICTRL). . . . 1-14
1.4.2.2 Interrupt Other Processor Register (INOP). . . 1-15
1.5 IBOX MICROCODE VISIBLE ONLY REGISTERS. 1-16
1.5.1 Clear Interrupt Other Processor (CIOP) 1-16
1.5.2 IBox Error Register (IBER) 1-16
1.5.2.1 IBER Usage 1-16
1.5.2.2 IBER Bits <7:0>. 1-17
1.5.2.3 IBER Bits <11:08>. 1-17
1.5.3 Clear Error Register (CER) 1-19

CHAPTER 2 MICROCODE OVERVIEW AND PIPELINE CONCEPTS

2.1 CHAPTER SCOPE. 2-1
2.2 VAX 8800 MAIN CONTROL STORE OVERVIEW 2-1
2.2.1 Microcode Size and Allocation. 2-1
2.2.2 Microcode File Structure 2-1
2.2.3 Microcode Assembly 2-2
2.2.3.1 Other Loadable Binary Files. 2-3
2.2.4 Microword Format 2-3
2.2.4.1 Field Naming Convention. 2-3
2.2.4.2 Field Functionality. 2-3
2.2.5 Microcode Definition Files 2-11
2.2.5.1 Field Definition File - DEFIN.MIC. 2-11
2.2.5.2 Macrodefinition File - MACRO.MIC 2-13
2.2.6 Microcode Related Documentation. 2-15
2.3 MICROCODE PIPELINING CONCEPTS. 2-20
2.3.1 Pipelining Rationale 2-20
2.3.2 Pipelined Versus Nonpipelined Machines 2-20
2.3.2.1 Performance Factors. 2-22
2.4 VAX 8800 PIPELINE CHARACTERISTICS. 2-23
2.4.1 CPU Clock Cycle. 2-23
2.4.2 CPU Hardware Design. 2-23
2.4.3 Relationship Between Microcycles and CPU Functions. 2-25
2.4.4 IB Decode Cycle. 2-25
2.4.5 Canonical Time States. 2-27
2.4.5.1 Definition of a Canonical Time State 2-27
2.4.5.2 Overlapping Time States. 2-27
2.4.6 Time State Events. 2-29

CHAPTER 3 IBOX FUNCTIONAL DESCRIPTION

3.1 CHAPTER SCOPE. 3-1
3.2 CONTROL STORE LOGIC. 3-2
3.2.1 Control Store RAM Segments 3-2
3.2.2 Control Store RAM Addressing 3-5
3.2.3 Control Store RAM Data Latches 3-5
3.2.4 Loading the Control Store RAMs 3-6
3.2.4.1 Load Control Store Microaddress. 3-6
3.2.4.2 Write Data to Selected Address 3-7
3.3 MICROSEQUENCING. 3-9
3.3.1 Microsequencer Hardware. 3-9
3.3.1.1 Microbranch Slice (UBRS) MCAs. 3-9
3.3.1.2 Microtrap (UTRP) MCA 3-9
3.3.1.3 Microstack 3-9
3.3.2 Normal Microcode Flow. 3-11
3.3.3 IB Decoder Supplied Microaddress 3-11
3.3.4 Microbranching 3-11
3.3.4.1 Microbranch Conditions 3-13
3.3.4.2 Microbranch Latency 3-19
3.3.5 Microsubroutine Calls and Returns. 3-21
3.3.5.1 Normal Microstack Operation. 3-21
3.3.5.2 Microsubroutine Calls. 3-21

3.3.5.3 Microsubroutine Returns. 3-23
3.3.6 Microtraps 3-23
3.3.6.1 Microtrap Servicing. 3-25
3.3.6.2 Disabling Microtraps 3-30
3.3.7 Console Supplied Microaddresses. 3-30
3.4 MACROINSTRUCTION DECODING. 3-32
3.4.1 Initializing the IB (IB Flush) 3-32
3.4.1.1 Full IB Flush. 3-32
3.4.1.2 IB Flush Logic 3-34
3.4.1.3 Partial IB Flush 3-38
3.4.2 I-Stream Prefetching 3-39
3.4.2.1 General Description. 3-39
3.4.2.2 Refilling Cache. 3-39
3.4.3 Loading the IB 3-40
3.4.3.1 IB Write Control 3-40
3.4.3.2 Cache Monitor Logic. 3-42
3.4.3.3 IB Full Logic. 3-43
3.4.3.4 IB Load Example. 3-43
3.4.4 Reading The IB 3-45
3.4.4.1 Pipeline Timing. 3-45
3.4.4.2 IB Read Ports. 3-46
3.4.4.3 IB Data Aligner. 3-51
3.4.4.4 IB Data Formatter and Data Scrambler 3-60
3.4.4.5 IB Read Example. 3-66
3.4.5 IB Manager Operations. 3-73
3.4.5.1 IB Read Address Logic. 3-73
3.4.5.2 Opcode Watcher Logic 3-76
3.4.5.3 Specifier Size Logic 3-78
3.4.5.4 Checking TEMPINC <2:0> Validity. 3-81
3.4.5.5 Decoder Stall. 3-82
3.4.5.6 Modifying the IB Pointer 3-82
3.4.5.7 IB Read Address Logic Control Signals. . . . 3-85
3.4.5.8 Computing Number of IB Longwords Consumed. . 3-91
3.4.6 Instruction Decoder Operation. 3-93
3.4.6.1 Pipeline Timing Considerations 3-93
3.4.6.2 Operand Specifier Entry Point Addresses. . . 3-96
3.4.6.3 Opcode Entry Point Microaddresses. 3-104
3.4.6.4 Special Microaddresses 3-111
3.4.6.5 IBST MCA Signals Related to Instruction Decoding 3-118
3.4.6.6 Decoder RAM (DRAM) 3-121
3.4.7 Optimized Instructions 3-128
3.4.7.1 Simple Move Instructions 3-128
3.4.7.2 Simple Branch Instructions 3-129
3.5 MACROBRANCH INSTRUCTIONS 3-129
3.5.1 Branch Instruction Basics. 3-129
3.5.2 Branch Instruction Classes 3-130
3.5.3 Unconditional Branches 3-130
3.5.4 Short Conditional Branches 3-135
3.5.5 Long Conditional Branches. 3-141
3.5.6 Condition Code and Macro Branch Logic. 3-147

3.6 SPECIAL REGISTER ADDRESSING. 3-150
3.6.1 RNUM1 and RNUM2 Registers. 3-150
3.6.2 RLOG Register. 3-150
3.6.3 MDNUM Register 3-152
3.7 INTERRUPTS 3-153
3.7.1 Interrupt Requests 3-153
3.7.2 Interrupt Servicing. 3-153
3.8 CONSOLE GATEWAY CONTROL. 3-158
3.8.1 Loading Control Store and Decoder RAMs 3-158
3.8.2 Starting the Micromachine. 3-160
3.8.3 Data Transfer with Console Resident IPRs . . . 3-160
3.8.4 Breakpoint Microtrap 3-160
3.8.5 Console Data Parity Check. 3-161

FIGURES

No. Title Page

1-1 IBox Block Diagram 1-3
1-2 NMI Interrupt Control (NICTRL) Register Bit Map. 1-14
1-3 Interrupt Other Processor (INOP) Register Bit Map. 1-15
1-4 IBox Error Register (IBER) Bit Map 1-18

2-1 Microword Bit Format 2-4
2-2 Sample Microword Field Definiton - I_APORT Field. 2-11
2-3 Basic Time State Diagram - Nonpipelined CPU. . . 2-21
2-4 Basic Time State Diagram - Pipelined CPU 2-21
2-5 Basic CPU Timing 2-24
2-6 Microcycles/CPU Functions. 2-26
2-7 IB Decode Cycle. 2-28
2-8 Canonical Time States. 2-28
2-9 VAX 8800 Pipeline Time State Diagram 2-30

3-1 Control Store Logic Simplified Block Diagram . . . 3-3
3-2 Microaddress Bit Slices for Micromatch Register Loading. 3-7
3-3 Control Store RAM Load Path. 3-8
3-4 Microsequencer Logic 3-10
3-5 Normal INEXT Field Addressing. 3-12
3-6 Microbranch Condition Selection. 3-14
3-7 Microbranch Latency. 3-20
3-8 Microstack Operation 3-22
3-9 Microtrap Servicing. 3-26
3-10 Microtrap Latency 3-27
3-11 Console Supplied Microaddress. 3-31
3-12 Instruction Buffer Logic 3-33
3-13 IB Flush Logic 3-35
3-14 IB Load Logic. 3-41

3-15 I-Stream Data Entering the IB. 3-44
3-16 IB Memory Unit Contents for MOVL Example 3-47
3-17 IB Read Port Example - Part 1. 3-48
3-18 IB Read Port Example - Part 2. 3-49
3-19 IB Data Aligner Muxes. 3-51
3-20 IB Data Aligner Output Example - Part 1. 3-54
3-21 IB Data Aligner Output Example - Part 2. 3-55
3-22 Opcode Mux Sources 3-57
3-23 IB Data Formatter and Data Scrambler Logic . . . 3-61
3-24 IB Read Example. 3-69
3-25 PCNC MCA Block Diagram 3-74
3-26 Simplified IB Read Address Logic 3-83
3-27 Instruction Decode Logic 3-94
3-28 Operand Specifier Entry Address Format 3-96
3-29 Opcode Entry Address Format. 3-104
3-30 Special Microaddress Format. 3-111
3-31 Special Address Encoder Logic. 3-112
3-32 Decoder RAM Read Address Format. 3-121
3-33 Decoder RAM Output Signals 3-123
3-34 Pipeline State for a BRB Instruction 3-131
3-35 Pipeline State for a Successful BEQL Instruction. 3-140
3-36 Pipeline State for a Successful AOBLEQ Instruction. 3-145
3-37 Condition Code and Macro Branch Logic. 3-148
3-38 File Address Slice MCAs. 3-151
3-39 Interrupt Logic Simplified Block Diagram 3-155
3-40 Gateway Control Logic. 3-159

TABLES

No. Title Page

1-1 Microcode Features 1-9
1-2 IBox Resident IPRs 1-13
1-3 NICTRL Register Bit Descriptions 1-14
1-4 INOP Register Bit Description. 1-15
1-5 IBER Bit Descriptions. 1-10

2-1 VAX 8800 Microcode Files 2-2
2-2 Microword Field Definitions. 2-6
2-3 Macroexpression Classes. 2-15
2-4 Sample Register Transfer Macros. 2-16
2-5 Sample Cache Command Macros. 2-17
2-6 Sample CREG/IREG Macros. 2-17
2-7 Sample Microbranch Macros. 2-18
2-8 Sample Miscellaneous Macros. 2-19
2-9 Pipelined/Nonpipelined CPU Comparisons 2-22
2-10 Pipeline Time States/CPU Events. 2-31

3-1 Control Store RAM Segment Functionality. 3-2
3-2 Next Microaddress Sources. 3-5
3-3 IBRTYPE/IBRMASK Microword Field Relationship . . 3-15
3-4 Microbranch Conditions 3-16
3-5 Special Microbranch Condition Bit Usage. 3-18
3-6 Microtrap Conditions and Vectors 3-24
3-7 Machine Check Microtrap Conditions 3-25
3-8 IBST and PCNC MCA Outputs After an IB Flush. . . 3-37
3-9 IB Flush Relative State Changes - IBST and PCNC MCAs 3-38
3-10 IB Read Address/IB Read Port Source. 3-46
3-11 Data Aligner Control Signals/Data Selection. . . 3-52
3-12 IB Data Format Control Signals/Functions 3-62
3-13 IB Format Control Signals/Specifier Data Type. . 3-63
3-14 IB Data Formatter and Data Scrambler Output. . . 3-67
3-15 Floating Point Short Literal Formats 3-68
3-16 Specifier Size Logic Control Signals 3-79
3-17 Slow Spec Size <2:0> Values. 3-80
3-18 IB Pointer Source. 3-84
3-19 Operand Specifier Entry Address Bit Descriptions 3-97
3-20 Operand Data Size/Access Type Correlation. . . . 3-98
3-21 Operand Specifier Entry Address Symbolic Labels 3-100
3-22 Opcode Entry Address Bit Descriptions. 3-105
3-23 Special Microaddress Conditions. 3-114
3-24 Special Conditions Serviced During IB Decode Cycles 3-116
3-25 Decoder RAM Output Signal Descriptions 3-124
3-26 Execute Code for a BRB Instruction 3-132
3-27 Microword CTL.BRB.MEM Event Timing 3-134
3-28 IMISC Field Settings for Macrobranch Recipes . . 3-136
3-29 IMISC Field Settings for PSL Condition Code Recipes 3-143
3-30 Execute Code for a AOBLEQ Instruction. 3-146
3-31 IMISC Field Settings for State Flag Control. . . 3-149
3-32 Hardware Interrupt Priority Levels 3-154
3-33 Interrupt ID Codes/IPLs. 3-157

EXAMPLE

No. Title Page

2-1 Sample Field Value Assignments - I_APORT 2-12

SECTION 7 EXECUTION BOX LOGIC (EBOX)

CHAPTER 1 INTRODUCTION

1.1 GENERAL. 1-1
1.1.1 EBox Organization. 1-2
1.1.2 EBox Operators 1-4
1.1.2.1 Main ALU 1-4
1.1.2.2 Cache Data Path (CDP) and Bus Watcher/Decoder (BWD). 1-4
1.2 SLICE MODULE (SLC1/SLC0) FUNCTIONS 1-5
1.2.1 Parity Generator/Checker (PAR) 1-5
1.2.2 Register File (RGF). 1-6
1.2.3 Slow Data File (SDF) 1-6
1.2.4 Program Counter (PC) Subsystem 1-7
1.2.5 Cache Data Path (CDP). 1-7
1.2.6 Main Arithmetic Logic Unit (Main ALU). 1-8
1.2.7 Bus Watcher/Decoder (BWD). 1-8
1.3 SHIFTER MODULE (SHR) FUNCTIONS 1-9
1.3.1 Shifter (SHF). 1-9
1.3.1.1 Integer Data 1-9
1.3.1.2 Floating-Point Data. 1-9
1.3.1.3 Decimal String Data. 1-10
1.3.2 Floating-Point (FP) Support. 1-10
1.3.2.1 Priority Encoder (PEN) 1-10
1.3.2.2 Shift ALU (SALU) 1-11
1.3.2.3 Exponent ALU (XALU). 1-11
1.3.3 Multiplier/Divider (MULT). 1-11
1.4 EBOX REGISTERS 1-12
1.4.1 P0LR, P1LR, and SLR Internal Bit Formats 1-13
1.4.2 VAX 8800-Specific Registers. 1-13
1.4.2.1 Machine Check Status Register (MCSTS). 1-13
1.4.2.2 System Identification (SID) Register 1-14
1.4.2.3 Revision Registers (REVR1 and REVR2) 1-15
1.4.2.4 EBox Parity Error Register (EBER). 1-17

CHAPTER 2 FUNCTIONAL DESCRIPTION

2.1 GENERAL. 2-1
2.2 SLICE MODULE (SLC1/SLC0) DESCRIPTION 2-1
2.2.1 Parity Generator/Checker (PAR) 2-1
2.2.1.1 Parity Generator 2-8
2.2.1.2 Parity Checker 2-8
2.2.1.3 EBox Parity Error Register (EBER). 2-11
2.2.1.4 Carry Save Logic 2-12
2.2.2 Register File (RGF). 2-14
2.2.2.1 Floating-Point Shuffle (FPS) 2-14
2.2.2.2 Memory Data (MD) Registers 2-16
2.2.2.3 Traps and Stalls 2-16
2.2.3 Slow Data File (SDF) 2-18
2.2.3.1 Writes 2-19
2.2.3.2 Reads. 2-19
2.2.3.3 Stalls and Traps 2-19

2.2.4 Program Counter (PC) Subsystem 2-22
2.2.4.1 PC VA FA Multiplexer 2-22
2.2.4.2 Virtual Address (VA) File. 2-22
2.2.4.3 Trap Shadow Logic. 2-23
2.2.4.4 Program Counter (PC) 2-24
2.2.5 Cache Data Path (CDP). 2-31
2.2.5.1 Cache Data Buffer (CDBF) 2-31
2.2.5.2 Cache Data Store (CDS) 2-33
2.2.6 Main Arithmetic Logic Unit (Main ALU). 2-41
2.2.6.1 ALU First Half (ALF) 2-41
2.2.6.2 Main ALU Functions 2-44
2.3 SHIFTER MODULE (SHR) DESCRIPTION 2-51
2.3.1 Shifter (SHF). 2-51
2.3.1.1 Shift Count Bus. 2-55
2.3.1.2 General Function Selection 2-55
2.3.1.3 Logical Shift or Rotate. 2-58
2.3.1.4 Arithmetic Shift 2-58
2.3.1.5 Decimal String Conversion. 2-58
2.3.2 Floating-Point (FP) Support. 2-60
2.3.2.1 Priority Encoder (PEN) MCA 2-62
2.3.2.2 Shift ALU (SALU) 2-70
2.3.2.3 Exponent ALU (XALU). 2-78
2.3.3 Multiplier/Divider (MULT). 2-86
2.3.3.1 Data Interface Signals 2-86
2.3.3.2 Carry and Control Signals. 2-86
2.3.3.3 Logical and Arithmetic Functions 2-86
2.3.3.4 Multiplier Operation 2-95
2.3.3.5 Divider Operation. 2-96

FIGURES

No. Title Page

1-1 VAX 8800 CPU Kernel Block Diagram. 1-1
1-2 Execution Unit (EBox) Block Diagram. 1-3
1-3 Machine Check Status Register (MCSTS). 1-13
1-4 System Identification (SID) Register 1-14
1-5 Revision Register 1 (REVR1). 1-15
1-6 Revision Register 2 (REVR2). 1-16

2-1 Slice Module (SLC1/SLC0) Block Diagram 2-2
2-2 Parity Generator/Checker (PAR) Block Diagram . . . 2-3
2-3 EBox Parity Error Register (EBER). 2-11
2-4 Register File (RGF) Block Diagram. 2-15
2-5 Slow Data File (SDF) Block Diagram 2-19
2-6 Program Counter (PC) Subsystem Block Diagram . . 2-23
2-7 Cache Data Path (CDP) Block Diagram. 2-32
2-8 Main Arithmetic Logic Unit (Main ALU) Block Diagram. 2-43
2-9 Shifter Module (SHR) Block Diagram 2-52
2-10 Shift MCA (SHFT) Logic and Gating Block Diagram. 2-53

2-11 Shift Control MCA (SHC) Block Diagram. 2-54
2-12 Shift Count Bus Signal and Gating Block Diagram. 2-55
2-13 VAX-11 Floating-Point Formats. 2-61
2-14 Priority Encoder (PEN) Block Diagram 2-64
2-15 INMUX Mapping of the BPORT Input Data. 2-65
2-16 Shift ALU (SALU) Block Diagram 2-73
2-17 Exponent ALU (XALU) Block Diagram. 2-79
2-18 Multiplier/Divider (MULT) Block Diagram. 2-88

TABLES

No. Title Page

1-1 Privileged IPRs Maintained by the EBox 1-12
1-2 P0LR, P1LR, and SLR Internal Formats 1-13
1-3 Machine Check Status Register (MCSTS) Bit Descriptions 1-14
1-4 System Identification (SID) Register Bit Field Descriptions 1-15
1-5 Revision Register 1 (REVR1) Bit Field Descriptions 1-16
1-6 Revision Register 2 (REVR2) Bit Field Descriptions 1-17

2-1 Parity Generator/Checker (PAR) Signal Descriptions 2-3
2-2 A-Side Port Control. 2-10
2-3 B-Side Port Control. 2-10
2-4 Keepgoing Conditions for A CD PAR<3,1> 2-10
2-5 E_SHFT<4:0> Control of the EBox Parity Error Register (EBER). 2-12
2-6 E_ALUCI<1:0> Control of the ALU Carry-In Source 2-13
2-7 Register File (RGF) Address Allocation 2-16
2-8 Register File (RGF) Signal Descriptions. 2-17
2-9 Slow Data File (SDF) Signal Descriptions 2-20
2-10 Program Counter (PC) Subsystem Signal Descriptions 2-27
2-11 E_VAWRT and I_APORT<7:6> Control of PC VA FA Multiplexer Input Selection. 2-30
2-12 PC Multiplexer Input Selection PC Multiplexer Select Signals 2-30
2-13 Cache Data Buffer (CDBF) Signal Description. . . 2-34
2-14 Cache Data Store (CDS) Signal Description. . . . 2-38
2-15 CDS Output Multiplexer Control for Each Slice. . 2-40
2-16 ALU First Half (ALF) Signal Descriptions 2-45
2-17 ALU Second Half (ALS) Signal Descriptions. . . . 2-48
2-18 A-Side Select (ASEL) Input Control Signals . . . 2-49
2-19 B-Side Select (BSEL) Input Control Signals . . . 2-49
2-20 Keepgoing/Stall Conditions 2-49
2-21 EALU<5:0> Field Control of the Main ALU. 2-50

2-22 Shift Count Bus Signals and Source 2-55
2-23 ESHFT<4:0> Field Selection of Shifter (SHF) MCA Logic Functions. 2-56
2-24 ESHFTSEL Selection of a Result Output to the BP Bus. 2-57
2-25 EFPFORMAT<1:0> Field Control of Decimal String Data Conversions. 2-59
2-26 EPEFUNC Field Selection of PEN Functions 2-65
2-27 Priority Encoder (PE) Results Passed to the Shift Count Bus 2-67
2-28 Increment Multiplexer Data (INCD) Selection to the Incrementer (INCR). 2-68
2-29 Sticky Bit Logic Input and Test Selection. . . . 2-68
2-30 G<1:0> Guard Bit Input Selection 2-69
2-31 Round Bit R<1:0> Input Selection 2-69
2-32 SALU and XALU Control Signals from the Microcode 2-71
2-33 ESXALUFN Field Control of the SALU Functions . . 2-72
2-34 Resulting Sign of the Fraction 2-74
2-35 SALU Selection of the APORT and BPORT Inputs . . 2-74
2-36 A-Latched Condition Code Inputs to the Branch Multiplexer. 2-76
2-37 Microbranch Condition Code Description 2-77
2-38 ESXALUFN<5:3) Control of the General XALU Functions. 2-80
2-39 XALU Functions with E_SXALUFN<5:3> Equal to 000 2-81
2-40 XALU Functions with ESXALUFN<5:3> Not Equal to 000 2-81
2-41 XALU Condition Code (XALUCC) Tests 2-82
2-42 M1 Inputs Passed to M3 2-83
2-43 M3 Inputs Passed to the Adder B-side 2-84
2-44 M2 Outputs to M6 or the XREG 2-85
2-45 M2 Data Passed to the BP Bus by M6 2-85
2-46 Multiplier/Divider (MULT) Control Signals from the Microcode 2-90
2-47 E_MULDIV Field Control of the MULT Functions . . 2-91
2-48 MULT Logic Signal Port Function Description. . . 2-93

SECTION 8 CACHE BOX LOGIC (CBOX)

CHAPTER 1 INTRODUCTION

1.1 CACHE BOX SYSTEM DESCRIPTION 1-1
1.2 CBOX OPERATION 1-5
1.2.1 CBox Cycles. 1-6
1.2.1.1 Quiescent State. 1-7
1.2.1.2 Read Cycle 1-8
1.2.1.3 Write Cycles 1-8
1.2.1.4 The PIBA 1-9

1.2.1.5 TB Cycles. 1-9
1.2.1.6 Refill Operation 1-10
1.2.1.7 Invalidate Cycle 1-10
1.2.2 CBox Stalls. 1-11

CHAPTER 2 FUNCTIONAL DESCRIPTION

2.1 CBOX SUBSYSTEMS DESCRIPTION. 2-1
2.1.1 Translation Buffer 2-1
2.1.1.1 VA Latch 2-4
2.1.1.2 TB RAM 2-5
2.1.1.3 TB Match MCA 2-8
2.1.1.4 TB RAM Bypass. 2-10
2.1.1.5 PA Latch 2-14
2.2 CBOX SUBSYSTEMS DESCRIPTION. 2-19
2.2.1 Cache. 2-19
2.2.1.1 Cache Data Path Logic. 2-19
2.2.1.2 Cache Tag MCA. 2-25
2.2.1.3 Cache Match MCA. 2-25
2.2.1.4 Cache Control MCA. 2-27
2.2.1.5 MD Number MCA. 2-28
2.3 CBOX SUBSYSTEMS DESCRIPTION. 2-29
2.3.1 NMI Interface. 2-29
2.3.1.1 NMI Address/Data Slices. 2-31
2.3.1.2 NMI Out Control. 2-36
2.3.1.3 NMI In Control 2-42
2.3.1.4 NMI Arbitration/Acknowledgment 2-47
2.3.1.5 CBox NMI Registers 2-52

FIGURES

No. Title Page

1-1 CBox -- Block Diagram. 1-3
1-2 CBox Cycle Timing. 1-5

2-1 Translation Buffer -- Block Diagram. 2-2
2-2 Virtual Address Fields 2-4
2-3 TB -- Write Sequence Diagram 2-6
2-4 TB Match MCA -- Simplified Block Diagram 2-9
2-5 PABH MCA -- Simplified Block Diagram 2-12
2-6 PABL MCA -- Simplified Block Diagram 2-13
2-7 PA Latch -- Logic Diagram. 2-15
2-8 PA Latch Bit Routing -- Refill Cycle 2-17
2-9 PA Latch Bit Routing -- VA Reference 2-18
2-10 Cache -- Block Diagram 2-20
2-11 NMI Interface -- Block Diagram 2-30
2-12 NMI Address/Data Slices MCA -- Simplified Block Diagram. 2-32
2-13 NMI Out Control -- Simplified Block Diagram. . . . 2-37
2-14 Control Store Microword Format Diagram 2-41
2-15 NMI In Control -- Block Diagram. 2-42

2-16 NMI Arbitration/Acknowledgment Control -- Simplified Block Diagram 2-49
2-17 Timeout -- Flow Diagram. 2-51
2-18 CBox NMI Registers -- Location Diagram 2-52
2-19 Cache Register -- Bit Format 2-53
2-20 Cache Error Register Byte 2 -- Bit Format. . . . 2-55
2-21 Cache Error Register Byte 1 -- Bit Format. . . . 2-56
2-22 Cache Error Register Byte 0 -- Bit Format. . . . 2-57
2-23 NMI Fault/Status Register Byte 1 -- Bit Format 2-58
2-24 NMI Fault/Status Register Byte 0 -- Bit Format 2-60
2-25 NMI Error Address Register -- Bit Format 2-61
2-26 NMI Silo Byte 2 -- Bit Format. 2-63
2-27 NMI Silo Byte 1 -- Bit Format. 2-65
2-28 NMI Silo Byte 0 -- Bit Format. 2-66
2-29 Cache TAG Initialization Register -- Bit Format 2-67
2-30 Diagnostic ID Register -- Bit Format 2-68
2-33 Diagnostic Control Register -- Bit Format. . . . 2-69

TABLES

No. Title Page

1-1 CBox Cycles. 1-7

2-1 PROTection Field <03:00> Coding and Access Allowed. 2-7
2-2 TB Match MCA Operation Coding. 2-8
2-3 Cache Register - Bit Descriptions. 2-53
2-4 Cache Error Register - Byte 2 Bit Descriptions 2-55
2-5 Cache Error Register - Byte 1 Bit Descriptions 2-56
2-6 Cache Error Register - Byte 0 Bit Descriptions 2-57
2-7 NMI Fault/Status Register Byte 1 - Bit Descriptions 2-58
2-8 NMI Fault/Status Register Byte 0 - Bit Descriptions 2-60
2-9 NMI Error Address Register -- Bit Descriptions 2-61
2-10 NMI Silo Byte 2 -- Bit Descriptions. 2-64
2-11 NMI Silo Byte 1 -- Bit Descriptions. 2-65
2-12 NMI Silo Byte 0 -- Bit Descriptions. 2-66
2-13 Cache TAG Initialization Register -- Bit Descriptions 2-67
2-14 Diagnostic ID Register -- Bit Descriptions . . . 2-68
2-15 Diagnostic Control Register -- Bit Descriptions 2-69

SECTION 9
MEMORY SYSTEM (MBOX)

CHAPTER 1
INTRODUCTION

1.1 MANUAL SCOPE

This document provides a technical description of the VAX 8800 memory system (MBox) hardware. It does not cover memory system architecture or software. Use is made of system microcode and software in explaining hardware functions; however, this manual does not purport to be a complete or thorough treatment of software aspects of the memory system.

Chapter 1 provides the purpose and major features of the MBox. It provides an overview of the memory system and describes how the MBox performs its function within the VAX 8800 system.

Chapter 2 provides a description of the memory controller (MCL). Included in Chapter 2 is a description of the NMI signals that interface with the MCL, an overview of the functional areas that comprise the MCL, and a detailed description of the functional areas. The MCL's functional areas coincide with the organization of the engineering documentation. Thus, the detailed descriptions are correlated with the engineering prints, specifications, etc.

Chapter 3 provides a description of the four-megabyte memory array board (MAR4). Included in Chapter 3 is a description of the MAR4 array bus (NAB) and how all of the NAB signals function on the MAR4 array board. It also includes an overview of the MAR4 and a detailed description of the MAR4's functional areas. The MAR4's functional areas coincide with the organization of the engineering documentation in the same manner as the Chapter 2 functional areas.

An appendix is provided that explains the flow-diagram symbology used in the MBox technical description.

1.2 WRITING PHILOSOPHY

It will be helpful to the reader to be aware of some points that were followed in the writing of this document. These points are explained in the following paragraphs.

The detailed descriptions in Chapters 2 and 3 assume that the reader has read and understands the overall descriptions. The detailed descriptions are confined to the area being described. It is important that the reader has read and understands the overview in order to understand how the detailed functions relate to the overall operation.

The functional block diagrams use logical AND and OR symbols. It does not necessarily follow that a corresponding gate exists on the engineering circuit prints. The assertion of inputs A and B causing the assertion of output C may be represented on a block diagram by a single AND gate, yet the engineering drawing may show that several circuit stages are involved in the ANDing operation.

The signal mnemonics used in the illustrations are identical to the signal mnemonics used in the engineering logic prints except for "fan-out" line designations which have a number or letter designation (<X>) that specifies a fan-out of a given signal. Fan-out lines are functionally identical, hence the fan-out line designations have been omitted to prevent confusion with other signal designations.

The signal mnemonics do not contain the H or L designations found on the engineering drawings. Signals are referred to as being asserted or negated without regard to high or low states. This point must be kept in mind when using timing waveform diagrams. As the signals have no H or L designation, they are always shown above the reference line for an asserted condition and on the reference line for a negated condition.

Flow diagrams are used extensively throughout this document. Refer to Apendix A (Flow Diagram Symbols) for any question on the meaning of any of the flow diagram symbols.

It is to be noted that the flow diagrams are NOT timing diagrams; in some cases the assertion or negation of a signal may not occur exactly when shown in a flow diagram. A given signal may assert before it serves its purpose in the flow of events. In such a case, the signal would appear in the diagram at the point it is used rather than at the point it was asserted. The flow diagrams are meant to illustrate the purpose and functioning of the hardware; to do this, the signals must be discussed at the time they are performing their function. Where signal timing has a functional significance, separate timing diagrams are used. With this restriction noted, the flow diagrams provide a useful representation of what is happening by serving as a pictorial supplement to the written descriptions.

1.3 MBOX FUNCTIONS

Figure 1-1 is a simplified block diagram of the MBox. The MBox interfaces with the NMI where it always functions as a slave nexus. The MBox consists of a memory controller (MCL) and up to eight identical four-megabyte array boards (MAR4s). The MAR4s interface with the MCL via an array bus (NAB). The NAB has eight board-select lines (AMCL BRD SEL<7:0>) -- one to each array board -- which enable a MAR4 to execute a function. The other NAB lines shown in Figure 1-1 are common to all the MAR4s.

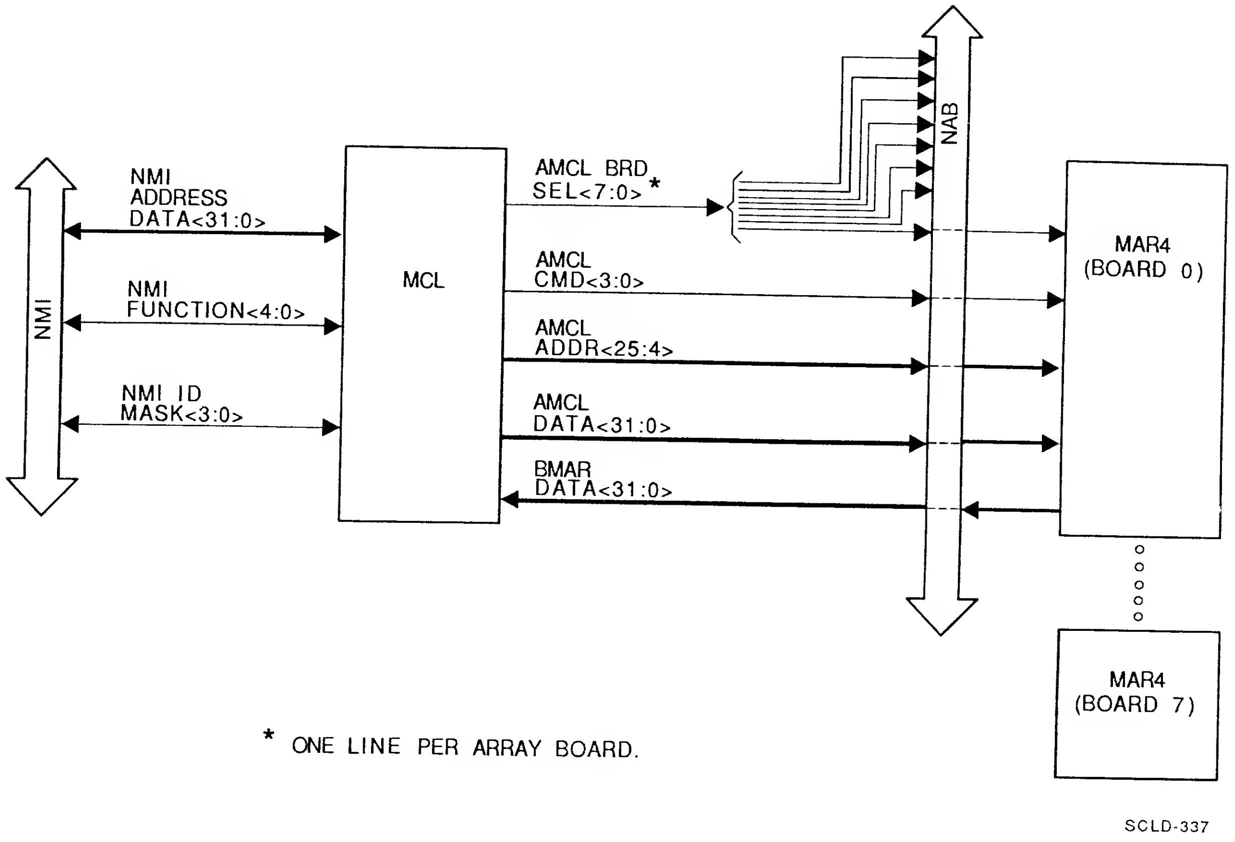


Figure 1-1 MBox Simplified Block Diagram

An NMI transaction is initiated by a command/address cycle wherein the NMI commander* places a function, an address, and its ID on the NMI. In a write transaction, the command/address cycle is followed by one or more data cycles in which the commander places the longword(s) of write data on the NMI. In a read transaction, the commander releases the NMI after the command/address cycle. When the MBox is ready to send the requested read data, it places the data on the NMI in read data cycle(s), along with the commander ID.

* Nexus that initiates a transaction on the NMI.

Figure 1-2 is a simplified flow diagram which illustrates the sequencing of read/write transactions. When a commander issues a command to memory, it places its ID code (NMI ID MASK<3:0>), the commanded function (NMI FUNCTION<4:0>), and the address (NMI ADDRESS DATA<31:0>) on the NMI during the command/address bus cycle. The MCL then:

- Stores the commander ID to be used if this is a read operation.
- Determines which MAR4 board is being selected from the NMI address, and asserts the select line (AMCL BRD SEL) for the selected array board.
- Transfers the NMI address to the MAR4 as AMCL ADDR<25:4> [address bits <3:2> become part of the 4-bit command (AMCL CMD<3:0>) and are used to select the array bank on the MAR4; address bits <1:0> are not used as all array addresses are longword aligned].
- Transfers the commanded function to the selected MAR4 by asserting AMCL CMD<3:0> on the NAB.

If a write operation is executing, the commander places the write data on the NMI address/data lines in the next (data) bus cycle. The MCL takes the write data from the NMI and transfers it to the NAB as AMCL DATA<31:0>. The selected MAR4 then writes the data into its memory arrays.

If a read operation is executing, the selected MAR4 supplies the addressed read data to the MCL as BMAR DATA<31:0>. The MCL places the read data on the NMI address/data lines along with an NMI function code specifying the data as good read data. The MCL also places the commander's ID code on the NMI ID mask<3:0> lines to identify to whom the read data belongs.

If a masked write operation is commanded, the commander places the masked write data on the NMI address/data lines, and the mask field on the ID/mask lines, during the data cycle. The masked write data is transferred to the MCL where it is held while the addressed location on the selected MAR4 is read. The BMAR DATA<31:0> read data is transferred to the MCL where it overwrites the stored write data according to the mask field. The new write data is then written into the MAR4 as AMCL DATA<31:0>.

1.4 MBOX OVERVIEW

Signals used in the MBox are prefixed with the phase of the clock to which they are referenced (A or B) and with the signal source. Thus, an "AMCL" prefix indicates the signal is referenced to the phase A clock and its source is the MCL. A "BMAR" prefix indicates the signal is referenced to the phase B clock and its source is the MAR4.

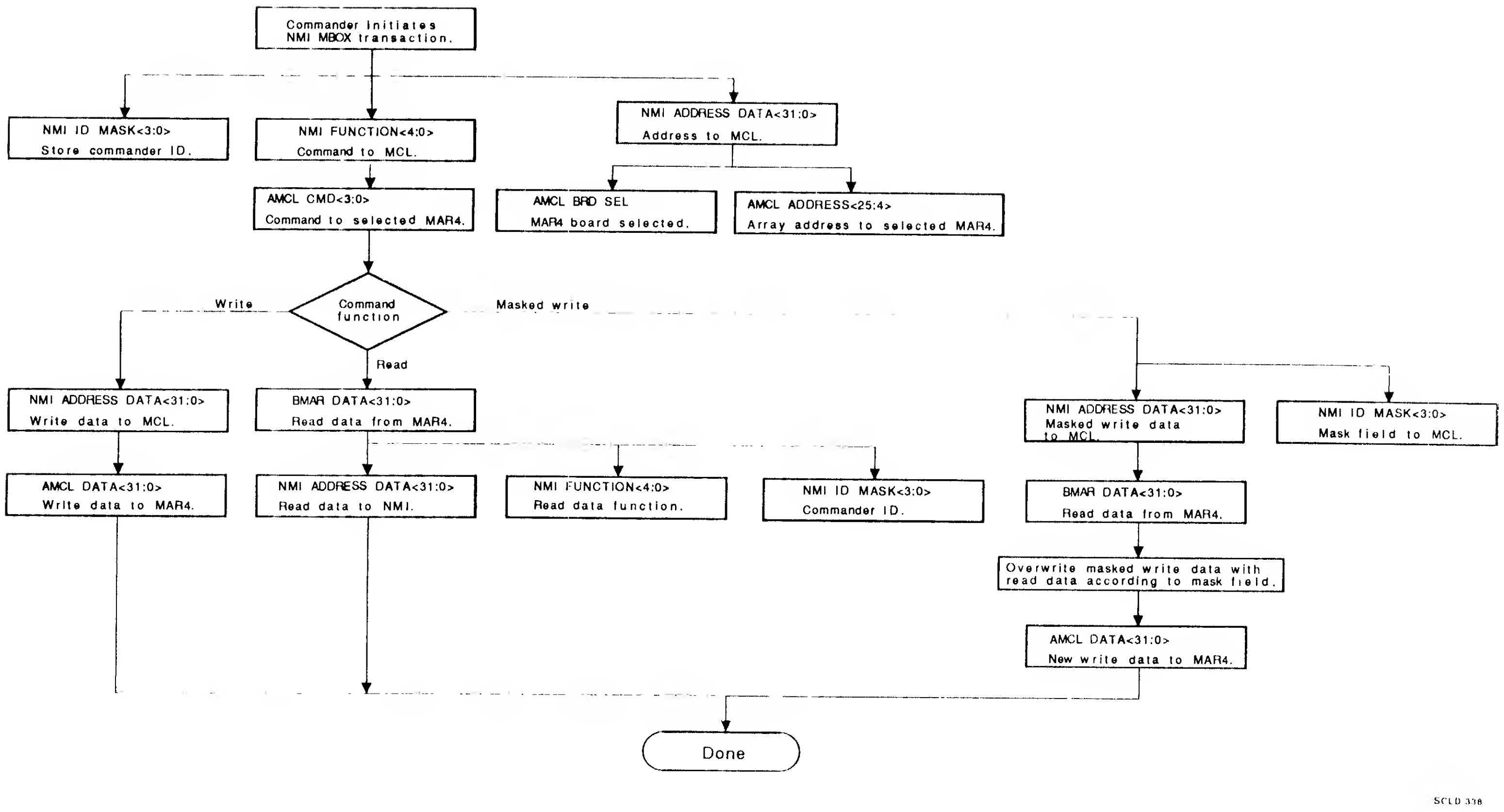

Commander initiates NMI MBOX transaction.
NMI ID MASK<3:0> Store commander ID.
NMI FUNCTION<4:0> Command to MCL.
NMI ADDRESS DATA<31:0> Address to MCL.
AMCL CMD<3:0> Command to selected MAR4.
AMCL BRD SEL MAR4 board selected.
AMCL ADDRESS<25:4> Array address to selected MAR4.
Write
Command function
Masked write
Read
NMI ADDRESS DATA<31:0> Write data to MCL.
BMAR DATA<31:0> Read data from MAR4.
NMI ADDRESS DATA<31:0> Masked write data to MCL.
NMI ID MASK<3:0> Mask field to MCL.
AMCL DATA<31:0> Write data to MAR4.
NMI ADDRESS DATA<31:0> Read data to NMI.
NMI FUNCTION<4:0> Read data function.
NMI ID MASK<3:0> Commander ID.
BMAR DATA<31:0> Read data from MAR4.
Overwrite masked write data with read data according to mask field.
AMCL DATA<31:0> New write data to MAR4.
Done
SCLD 338

Figure 1-2 MBox Read/Write Simplified Block Diagram

1.4.1 NMI Signals Used by the MBox

Signals on the NMI that interface with the MBox are shown in Table 1-1. Most of these signals are used in NMI/MBox transactions as described in Section 1.4.5.

1.4.2 MBox Operations

The operations supported by the MBox, and their corresponding NMI function codes, are listed in Table 1-2.

1.4.3 Octaword Sort-of-Write

A non-masked octaword write transaction is a "sort-of-write" operation in that the mask field associated with the longword is used to specify if the longword is to be written. The sort-of-write feature allows a nexus with longwords of data within an octaword address to use the octaword write function even though all the longwords are not to be written. This allows data to be transferred on the NMI in one large transfer, rather than in several slower transfers.

Non-masked longword writes are also sort-of-writes. The sort-of-write rationale does not apply to longword writes, however, due to logic standardization, the sort-of-write function is implemented for all non-masked write transactions.

1.4.4 Command Bus Cycles

Possible sizes of NMI transfers for read/write operations are:

- Longword writes
- Quadword writes (masked writes only)
- Octaword writes
- Longword reads
- Octaword reads
- Hexword reads

Sections 1.4.4.1 through 1.4.4.6 describe the NMI bus cycles that occur for the six transfers listed above. An NMI bus cycle is 50 ns.

Table 1-1 NMI Signals Used by the MBox

Signal	Direction	No. of Lines	Function
NMI ADDRESS DATA<31:0>	B	32	Address and data lines.
NMI DATA PARITY	B	1	Parity bit for address/data lines.
NMI FUNCTION<4:0>	B	5	Command function (Table 1-2).
NMI ID MASK<3:0>	B	4	Commander ID and mask code.
NMI FUNCT ID PARITY	B	1	Parity bit for function/ID/mask lines.
NMI CONFIRMATION<1:0>	U	2	MBox confirmation code (Table 1-4).
NMI MEMORY ARB	U	1	MBox arbitration request line.
NMI MCL BUS EN	U	1	NMI bus grant to MBox from CPU.
NMI MEMORY HOLD	U	1	Holds NMI for MBox after receiving bus grant.
NMI MEMORY BUSY	U	1	MBox busy line.
NMI MEM BUSY ARB	U	1	MBox busy line to arbitrator in CPU.
NMI LCPU MEM INTR	U	1	MBox interrupt line to left CPU.
NMI RCPU MEM INTR	U	1	MBox interrupt line to right CPU.
NMI FAULT DETECT<3:0>	U	4	Fault lines from other NMI nexus.
NMI FAULT	U	1	OR of MBox fault and fault lines from other NMI nexus.

B = bidirectional; U = unidirectional

Table 1-1 NMI Signals Used by the MBox (Cont)

Signal	Direction	No. of Lines	Function
NMI SLOW COUNT ENABLE	U	1	Increments MBox timeout counter.
NMI DC LO	U	1	Warning of dc power loss.
NMI BAT DC LO	U	1	Backup battery power low -- next power up will be a "cold start."
NMI SLOW MODE	U	1	Warning that clocks are about to stop.
NMI HARBINGER	U	1	Inhibits MBox clocks.
NMI UNJAM	U	1	Initializes MBox.
NMI F A CLK IN	U	1	Free running, phase A clock.
NMI F B CLK IN	U	1	Free running, phase B clock.

B = bidirectional; U = unidirectional.

Table 1-2 MBox Command Functions

(Hex)	Function <4	3 2 1 0>	Command
1 0	1	0 0 0 0	Read longword
1 2	1	0 0 1 0	Read octaword
1 3	1	0 0 1 1	Read hexword
1 4	1	0 1 0 0	Read longword interlocked
1 6	1	0 1 1 0	Read octaword interlocked
1 7	1	0 1 1 1	Read hexword interlocked
1 B	1	0 1 1	Write longword
1 F	1	1 1 1 1	Write octaword
1 8	1	1 0 0 0	Write masked longword
1 9	1	1 0 0 1	Write masked quadword
1 A	1	1 0 1 0	Write masked octaword
1 C	1	1 1 0 0	Write masked longword unlock
1 D	1	1 1 0 1	Write masked quadword unlock
1 E	1	1 1 1 0	Write masked octaword unlock

Table 1-2 MBox Command Functions (Cont)

(Hex)	Function <4	3 2 1 0>	Command
0 A	0	1 0 1 0	Read/return good data
0 E	0	1 1 1 0	Read/return bad data
0 8	0	1 0 0 0	Read/continue good data
0 C	0	1 1 0 0	Read/continue bad data
0 9	1	1 0 0 1	Write data

1.4.4.1 Write Longword Bus Cycles (Table 1-3) --

Table 1-3 Write Longword Bus Cycles

Cycle	Cmd/ Addr	Data	----
ADDRESS DATA<31:00>	Addr	Data	----
FUNCTION<4:0>	Write LW	Write data	----
ID MASK<3:0>	CMDR ID	Byte mask	----
CONFIRMATION<1:0>	----	----	OK

In the command/address bus cycle the commander places the 32-bit address, a five-bit function code, and a four-bit ID/mask code on the NMI. The function code specifies the operation as a longword write and further defines it as a masked or unlock function. The ID/mask code identifies the transaction commander.

In the data cycle, the commander places the 32-bit data longword, a write-data function code, and a byte mask on the NMI. The write-data function code identifies the data cycle as a write-data cycle and specifies the data on the address/data lines as write data. The byte mask is the byte mask for a masked write transaction, or a write flag for a non-masked write transaction (see Sort-of-Write, Section 1.4.3).

In the third bus cycle, the MBox places a 2-bit confirmation code on the NMI specifying its response to the commander. Confirmation codes occur in the third bus cycle for all NMI transactions (reads and writes). Four confirmation responses are possible as shown in Table 1-4.

Table 1-4 NMI Confirmation Codes

NMI Confirmation <1	0>	Confirmation State
0	0	No acknowledgement
0	1	Command accepted
1	0	Interlock busy
1	1	Memory busy

1.4.4.2 Write Quadword Bus Cycles (Table 1-5) --

Table 1-5 Write Quadword* Bus Cycles

Cycle	Cmd/ Addr	Data	Data
ADDRESS DATA<31:00>	Addr	Data	Data
FUNCTION<4:0>	Masked write QW	Write data	Write data
ID MASK<3:0>	CMDR ID	Byte mask	Byte mask
CONFIRMATION<1:0>	----	----	OK

* Masked writes only.

In the command/address bus cycle the commander places the address, the function code, and the ID code on the NMI. The function code specifies the transaction as a masked quadword write (a non-masked quadword write is not a valid NMI function) and if the transaction is a write unlock function. The ID code identifies the transaction commander.

In the two write-data cycles that follow, the commander places a data longword, a write-data cycle code, and the byte mask on the NMI. The write-data cycle code identifies the cycle as a write-data cycle and specifies the data on the address/data lines as write data.

The third bus cycle is the data cycle for the second longword of write data. The confirmation code is placed on the NMI during this cycle.

1.4.4.3 Write Octaword Bus Cycles (Table 1-6) --

Table 1-6 Write Octaword Bus Cycles

Cycle	Cmd/ Addr	Data	Data	Data	Data
ADDRESS DATA<31:00>	Addr	Data	Data	Data	Data
FUNCTION<4:0>	Write OW	Write data	Write data	Write data	Write data
ID MASK<3:0>	CMDR ID	Byte mask	Byte mask	Byte mask	Byte mask
CONFIRMATION<1:0>	----	----	OK	----	----

In the command/address bus cycle the commander places the address, the function code, and the ID code on the NMI. The function code specifies the transaction as an octaword write and if the transaction is a masked or unlock function. The ID code identifies the transaction commander.

In the four write-data cycles that follow, the commander places a data longword, a write-data cycle code, and a byte mask on the NMI. The write-data cycle code specifies the data on the address/data lines as write data. The byte mask is the byte mask for a masked write transaction, or a write flag for a non-masked, sort-of-write transaction.

The confirmation code is placed on the NMI in the third bus cycle of the transaction.

1.4.4.4 Read Longword Bus Cycles (Table 1-7) -- In the command/address bus cycle the commander places the address, the function code, and the ID code on the NMI. The function code specifies the transaction as a longword read, and if the transaction is an interlock function. The ID code identifies the transaction commander.

The MBox places the confirmation code on the NMI in the third bus cycle of the transaction.

After the MBox has extracted the requested data from the array board, it arbitrates for the NMI by asserting NMI MEMORY ARB. When the arbitration logic in the CPU grants the arbitration request, it asserts NMI MCL BUS EN as a bus grant to the MBox. The bus grant is issued in the same arbitration cycle in which the bus was won.

Table 1-7 Read Longword Bus Cycles

Cycle	Cmd/ Addr			Arb	Data
ADDRESS DATA<31:00>	Addr	--	--	--	Data
FUNCTION<4:0>	Read LW	--	--	--	Read Rtn
ID MASK<3:0>	CMDR ID	--	--	--	CMDR ID
CONFIRMATION<1:0>	--	--	OK	--	--
NMI MEMORY ARB	--	--	--	Arb	--
NMI MCL BUS EN	--	--	--	BG	--

NOTE
Table 1-7 illustrates the case of the MBox winning the NMI bus right away.

The bus-grant cycle is followed by the first data cycle in which the MCL places the longword of read data on the NMI address/data lines, a read return function code on the function lines, and the commander ID on the ID/mask lines. There are four possible read functions as seen in Table 1-2. These are:

- Read/return good data
- Read/return bad data
- Read/continue good data
- Read/continue bad data

The first longword of read data placed on the NMI has a "read return" function code specifying it as the first longword of read data. The following longwords (if any) have a "read continue" function code.

The MCL performs an ECC check on all read data and corrects any correctable (single-bit) errors found. If a non-correctable (double-bit) error is detected, the erroneous longword is placed on the NMI with a "bad-data" function code. Otherwise, the read data carries a "good-data" function code.

1.4.4.5 Read Octaword Bus Cycles (Table 1-8) --

Table 1-8 Read Octaword Bus Cycles

Cycle	Cmd/ Addr			Arb	Data	Data	Data	Data
ADDRESS DATA<31:00>	Addr	--	--	--	Data	Data	Data	Data
FUNCTION<4:0>	Read OW	--	--	--	Read rtn	Read cont	Read cont	Read cont
ID MASK<3:0>	CMDR ID	--	--	--	CMDR ID	CMDR ID	CMDR ID	CMDR ID
CONFIRMATION<1:0>	--	--	OK	--	--	--	--	--
NMI MEMORY ARB	--	--	--	Arb	--	--	--	--
NMI MCL BUS EN	--	--	--	BG	BG	BG	BG	--
NMI MEMORY HOLD	--	--	--	--	Hold	Hold	Hold	--

NOTE
Table 1-8 illustrates the case of the MBox winning the NMI bus right away.

In the command/address bus cycle the commander places the address, the function code, and the ID code on the NMI. The function code specifies the transaction as an octaword read, and if the transaction is an interlock function. The ID code identifies the transaction commander.

The MBox places the confirmation code on the NMI in the third bus cycle of the transaction.

After the MBox has extracted the requested data from the array board, it arbitrates for the NMI. When the arbitration logic in the CPU grants the arbitration request, it asserts NMI MCL BUS EN as a bus-grant to the MBox. The bus-grant holds the NMI for only one cycle. If the MCL needs the NMI for the next bus cycle, it asserts NMI MEMORY HOLD. The CPU arbitration logic treats NMI MEMORY HOLD as an arbitration request and keeps NMI MCL BUS EN asserted.

The first bus-grant cycle is followed by the first of the four data cycles. The MCL places the first longword of read data on the NMI along with the read function code (read/return) and the commander ID. The MCL also asserts NMI MEMORY HOLD on the first bus cycle and keeps it asserted for the next two bus cycles. This gives the NMI to the MCL for the four bus cycles needed to transfer the data octaword.

If a non-correctable error is detected in any of the read longwords, a "bad-data" function code is placed on the NMI with the longword containing the non-correctable error.

1.4.4.6 Read Hexword Bus Cycles (Table 1-9) -- In the command/address bus cycle the commander places the address, the function code, and the ID code on the NMI. The function code specifies the transaction as a hexword read, and if the transaction is an interlock function. The ID code identifies the transaction commander.

The MBox places the confirmation code on the NMI in the third bus cycle of the transaction.

After the MBox has extracted the requested data from the array board, it arbitrates for the NMI. When the MBox receives the NMI bus grant, it places the first longword of read data on the NMI along with the read- data function code, the commander ID, and the NMI MEMORY HOLD signal.

If a non-correctable error is detected in any of the read longwords, a "bad-data" function code is placed on the NMI with the longword containing the non-correctable error.

A hexword of read data is transferred to the NMI in two octaword transfers. The MBox requires that there be at least one bus cycle between the two octaword transfers; hence, it must re-arbitrate for the NMI for the second octaword transfer.

1.4.5 Memory Controller (MCL)

The MCL interfaces the memory array boards with the NMI by performing the following functions.

- Controls execution of the NMI commands on the array boards, such as implementing masked and interlocked functions.
- Converts quadword and octaword writes into longword writes for the array boards.*
- Converts hexword reads into octaword reads for the array boards.*
- Performs ECC checking of read data, correcting single-bit errors and flagging double-bit errors.

---

* The array boards perform only longword write, longword read, and octaword read operations.

Table 1-9 Read Hexword Bus Cycles

Cycle	Cmd/ Addr			Arb	Data	Data	Data	Data	Arb	Data	Data	Data	Data
ADDRESS DATA<31:0>	Addr	--	--	--	Data	Data	Data	Data	--	Data	Data	Data	Data
FUNCTION<4:0>	Read HW	--	--	--	Read rtn	Read cont	Read cont	Read cont	--	Read rtn	Read cont	Read cont	Read cont
ID MASK<3:0>	CMDR ID	--	--	--	CMDR ID	CMDR ID	CMDR ID	CMDR ID	--	CMDR ID	CMDR ID	CMDR ID	CMDR ID
CONFIRMATION <1:0>	--	--	OK	--	--	--	--	--	--	--	--	--	--
NMI MEMORY ARB	--	--	--	Arb	--	--	--	--	Arb	--	--	--	--
NMI MCL BUS EN	--	--	--	BG	BG	BG	BG	--	BG	BG	BG	BG	--
NMI MEMORY HOLD	--	--	--	--	Hold	Hold	Hold	--	--	Hold	Hold	Hold	--

NOTE
Table 1-9 illustrates the case of the MBox winning the NMI bus right away.

Figure 1-3 is a block diagram of the MBox (MCL, NAB, and MAR4). The MCL is divided into three sections; the DFA, the MDP, and the MRM, whose functions are listed below:

- The DFA interfaces the memory system to the NMI.
- The MDP (memory data path) routes the read/write data between the DFA and the array boards.
- The MRM interfaces the MCL commands to the array boards, and monitors array board status.

Signal lines on the NAB are common to all of the array boards except the following four signals.

- AMCL BRD SEL<7:0>
- AMCL READ BD SEL<7:0>
- BMAR DATA RDY DONE<7:0>
- BMAR SEND NO CMD<7:0>

These signals output the MRM on eight lines with one line connected to each array board slot.

1.4.5.1 **Command/Address Sequence** -- Figure 1-4 is a flow diagram of the command/address sequence (receiving an NMI command and accessing the array boards). Refer to it and to Figure 1-3 during the following discussion.

When a nexus sends a command to memory, it places the following signals on the NMI in the command/address cycle:

- NMI FUNCTION<4:0> -- the command function
- NMI ID MASK<3:0> -- the commander ID
- NMI FUNCT ID PARITY -- parity over the function and ID/mask fields
- NMI ADDRESS DATA<31:00> -- the memory address
- NMI DATA PARITY -- parity bit for the address/data field

The memory system may not be able to accept a command at this time due to its being busy processing other commands, or due to being write-locked. If this is the case, the DFA asserts a "busy" or "write-locked" confirmation code (NMI CONFIRMATION<1:0>) on the NMI (Table 1-4). The commander will then try again at a later time. If the DFA determines the command to be invalid, it asserts a "no acknowledgement" confirmation code. If the command is accepted by the MCL, a "command accepted" confirmation code is asserted.

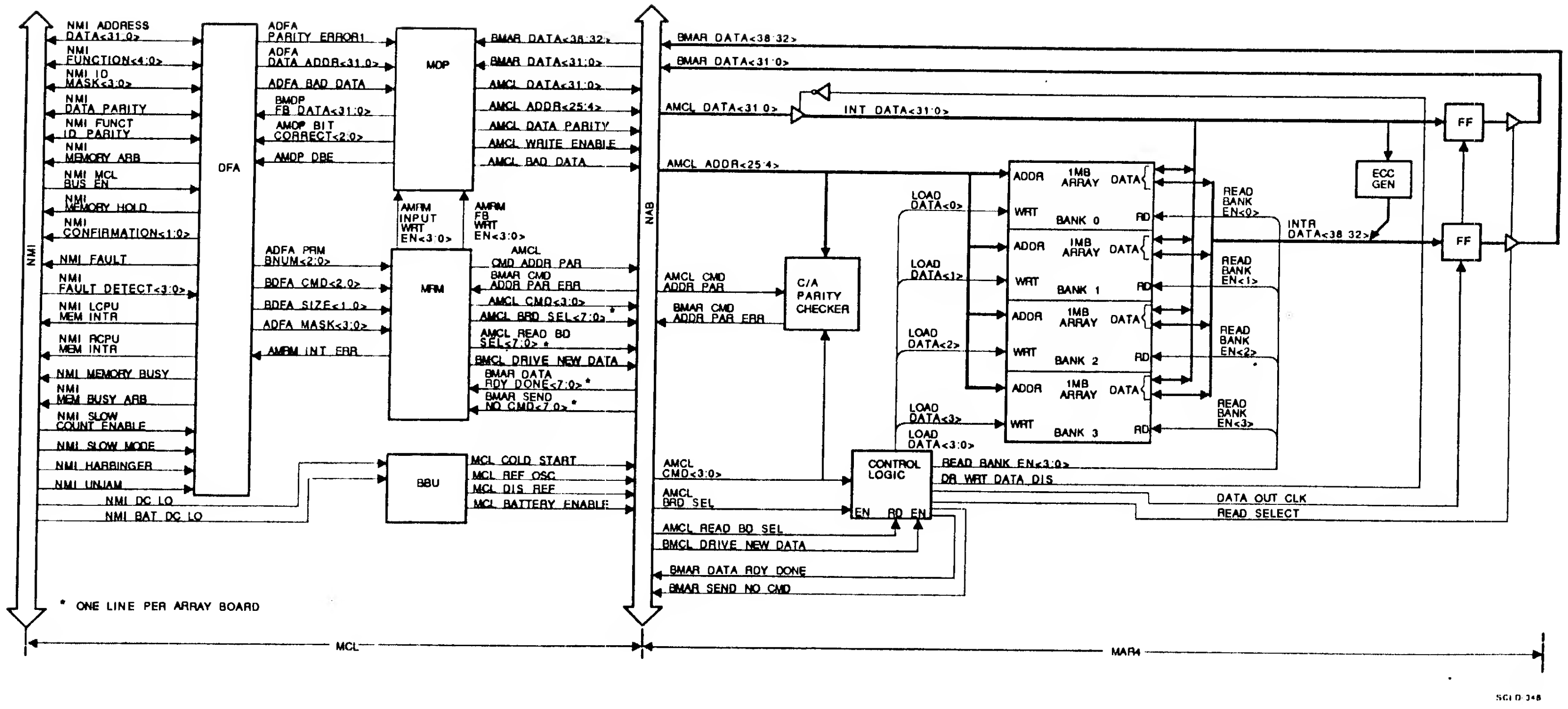


Figure 1-3 MBox Block Diagram

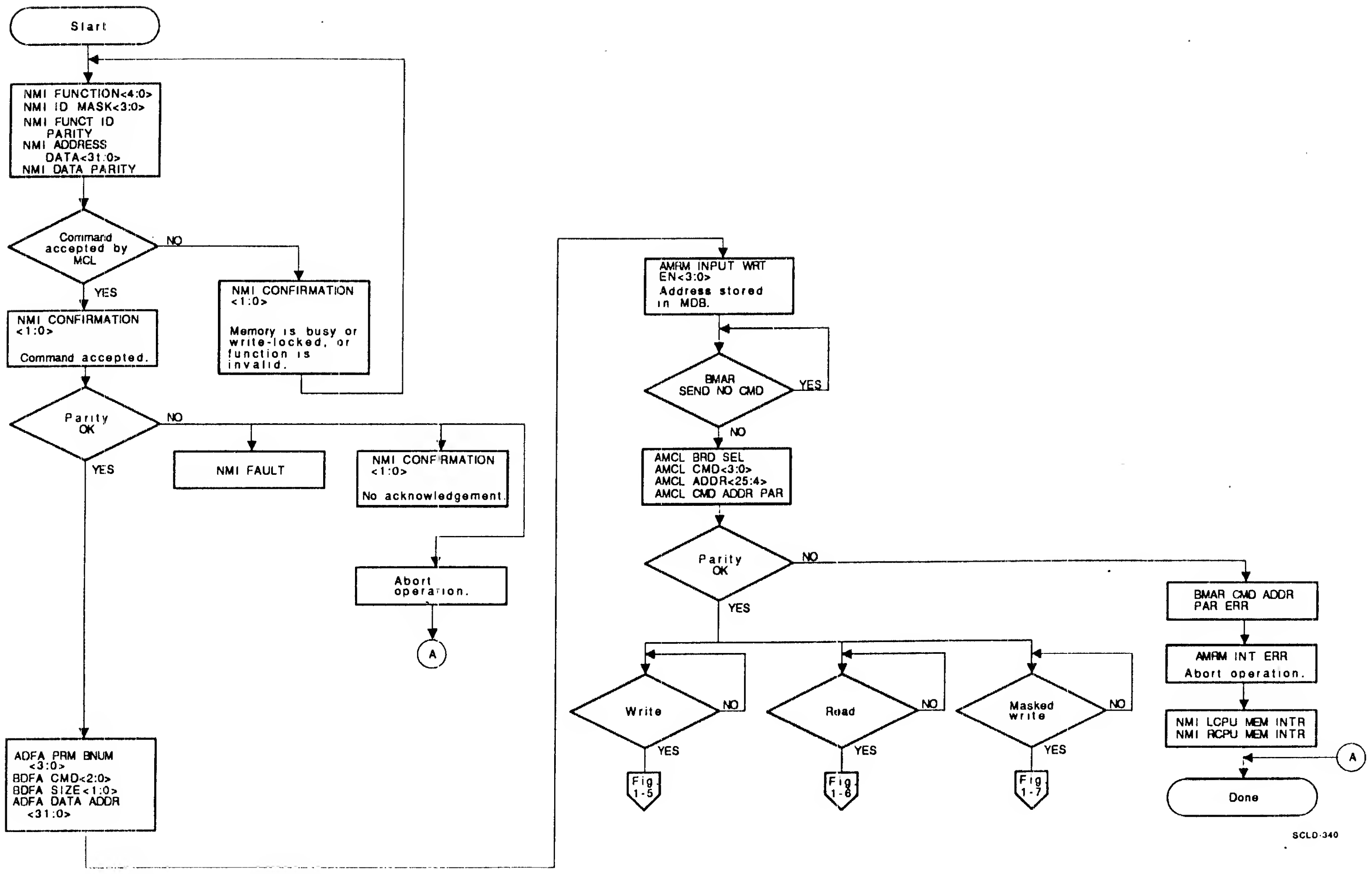


Figure 1-4 Command/Address Flow Diagram

Using the data parity bit and the function/ID/mask parity bit from the NMI, the DFA performs parity checks on the memory address and the function/ID field. If an error is detected, the DFA asserts NMI FAULT and a "no acknowledgement" confirmation code on the NMI, and the operation is aborted.

If no parity error is detected, the DFA processes the NMI data as follows:

- It extracts the array board number (ADFA PRM BNUM<2:0>) from the NMI address and sends it to the MRM.
- It extracts the type of operation (BDFA CMD<2:0>) and the size of the transfer (BDFA SIZE<1:0>) from the NMI function code, and sends them to the MRM.
- If a read operation is executing, it stores the commander ID for use when the read data is placed on the NMI,
- It transfers the address through the DFA to the MDP as ADFA DATA ADDR<31:0>. The address is stored in the MDP by AMRM INPUT WRT EN<3:0> from the MRM.

The MRM checks the BMAR SEND NO CMD line from the target array board. The board asserts the line if it is not able to accept a command at this time due to executing a previous read command or doing a refresh. If BMAR SEND NO CMD is false, the MRM:

- Enables the target array board by asserting the board's AMCL BRD SEL select line.
- Provides the four-bit command to the array board (AMCL CMD<3:0>). The command contains a read/write bit, a size bit, and two starting address bits. The starting address bits select the bank on the array board (array boards have four banks).
- Provides the address from the MDP as AMCL ADDR<25:4>.
- Provides a command/address parity bit generated by the MRM (AMCL CMD ADDR PAR).

The array board performs a parity check on the command/address fields and asserts BMAR CMD ADDR PAR ERR if an error is detected. The MRM responds to the parity error by asserting AMRM INT ERROR to abort the operation. AMRM INT ERROR is sent to the DFA which asserts the two CPU interrupt lines on the NMI (NMI LCPU MEM INTR and NMI RCPU MEM INTR).

If the parity checks OK, the command operation proceeds according to the type of operation in progress.

1.4.5.2 **Normal Write** -- Figure 1-5 is a flow diagram of a write operation data cycle and is used in conjunction with the block diagram (Figure 1-3).

In a write operation data cycle, the commander places the following signals on the NMI:

- NMI FUNCTION<4:0> -- the command function (write data cycle)
- NMI ID MASK<3:0> -- write flag
- FUNCT ID PARITY -- parity over the function and ID/mask fields
- NMI ADDRESS DATA<31:00> -- the write data longword
- NMI DATA PARITY -- parity bit for the write data

NMI FUNCTION<4:0> specifies a write-data function to the MCL. NMI ID MASK<3:0> is checked for the write flag that specifies that the write data is to be written into the memory arrays (sort-of-write operation, Section 1.4.3). An asserted flag causes AMCL WRITE ENABLE to be asserted to the array board.

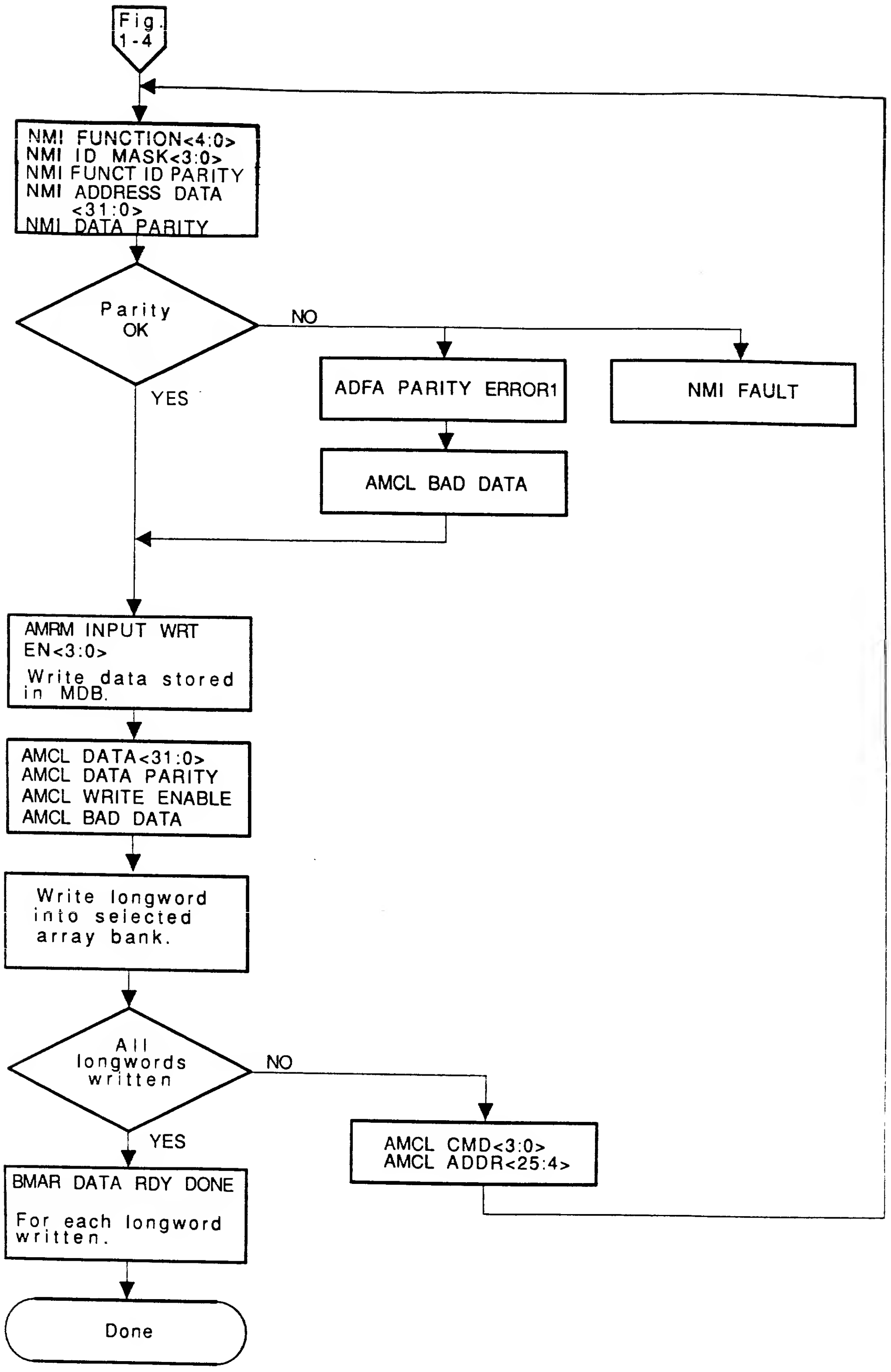
Using the data parity bit and the function/ID/mask parity bit from the NMI, the DFA performs parity checks on the write data and the function/ID/mask fields. If an error is detected, the DFA asserts NMI FAULT on the NMI and ADFA PARITY ERROR1 to the MDP. ADFA PARITY ERROR1 causes the MDP to set the bad-data bit (AMCL BAD DATA) on the NAB, however the write operation is allowed to execute. When the write data is written into the arrays, the asserted bad-data bit marks the data as bad. The data is written so that it can be analyzed later by maintenance diagnostics.

The write data (NMI ADDRESS DATA<31:0>) passes through the DFA to the MDP as ADFA DATA ADDR<31:0>. The write data is stored in the MDP by AMRM INPUT WRT EN<3:0> from the MRM.

The only write operation performed by the array boards is a longword write; hence, the MCL must convert quadword and octaword writes into a series of longword writes for the array board.

For a normal (non-masked) write operation, the array board does not assert its BMAR SEND NO CMD line due to an operation in progress, because only one array bank is being written (only longword writes are executed by the array boards). The other three banks are free to accept write commands.

The write data stored in the MDP is placed on the NAB as AMCL DATA<31:0>. The MDP also supplies a data parity bit, a write-enable bit, and a bad-data bit. The data parity bit allows the array board to make a parity check on the write data. If the array board detects a data parity error, the data is written into



SCLD-350

Figure 1-5 Write Data Cycle Flow Diagram

the arrays but is marked as bad data. The write-enable bit specifies whether the associated longword is to be written or not. Occasions when a longword is not to be written is for an unwanted longword during a sort-of-write operation (Section 1.4.3), or when a double- bit error is encountered during a masked write operation (Section 1.4.5.4). The bad-data bit (or a write-data parity error in the array board) specifies the associated longword as being bad. The bad-data bit is set in any data cycle in which the MCL detects a data parity error or a function/ID/MASK parity error (Section 1.4.5.2), or when the DFA detects missing write data and asserts ADFA BAD DATA to the MDP.

The longword of write data is then written into the selected array bank.

If the function being executed is a longword write, the board asserts BMAR DATA RDY DONE to inform the MRM that the array bank just written is ready to accept another command. The operation is now finished.

If the function being executed is a multiword write, the MRM re-asserts the longword write command and address for the next array bank. The next longword of write data from the NMI is processed and supplied to the array board along with the data-parity, write-enable, and bad-data bits. The process repeats until all the longwords are written.

BMAR DATA RDY DONE asserts for each longword of data that is written. The MRM does not have to received BMAR DATA RDY DONE from the first longword write command before it issues a second longword write command because the second longword will be going to the next array bank. In an NMI octaword write operation, the MRM will have initiated four longword write operations to the array board before it receives the first BMAR DATA RDY DONE. The MRM knows the first BMAR DATA RDY DONE signal asserted is for the first longword write operation.

1.4.5.3 Read -- Figure 1-6 is a flow diagram of a read operation and is used in conjunction with the block diagram (Figure 1-3).

Longword and octaword reads are the only read operations performed by the array boards. Hence, the MCL must convert a hexword read into two octaword reads for the array board.

For a read operation, the array board asserts it BMAR SEND NO CMD line to lock-up the board until the read operation is completed. A read of the array board executes in two segments: reading the arrays and taking the data. The two segments do not necessarily follow each other immediately, so when the first segment is initiated (reading the arrays) all new commands to the board must be locked out until the second segment is executed.

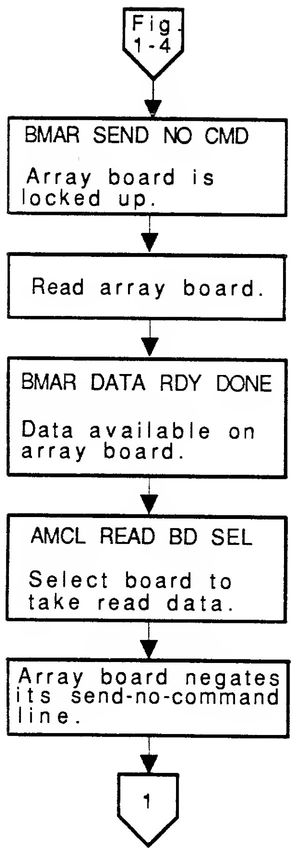


Fig 1-4
BMAR SEND NO CMD
Array board is locked up.
Read array board.
BMAR DATA RDY DONE
Data available on array board.
AMCL READ BD SEL
Select board to take read data.
Array board negates its send-no-command line.
1

SCLD-341A

Figure 1-6 Read Data Cycle Flow Diagram (Sheet 1 of 2)

The array board proceeds to read the data and then asserts BMAR DATA RDY DONE to inform the MRM that the arrays have been read and the data is available to be taken. When the MRM is ready to take the data, it asserts the board's AMCL READ BD SEL line to prepare the board to send the data. When the array board is read-selected by AMCL READ BD SEL, it negates its BMAR SEND NO CMD line. The MRM then asserts BMCL DRIVE NEW DATA to transfer the read data to the MDP. The read data consists of a longword of data (BMAR DATA<31:0>) and seven ECC check bits (BMAR DATA<38:32>).

The MDP performs an ECC check on the read data. If no ECC error is found, the data longword is sent to the DFA as feedback data (BMDP FB DATA<31:0>). The DFA generates a "good read data" function code and then proceeds to arbitrate for the NMI by asserting NMI MEMORY ARB. When the arbitration logic in the CPU grants the NMI to the

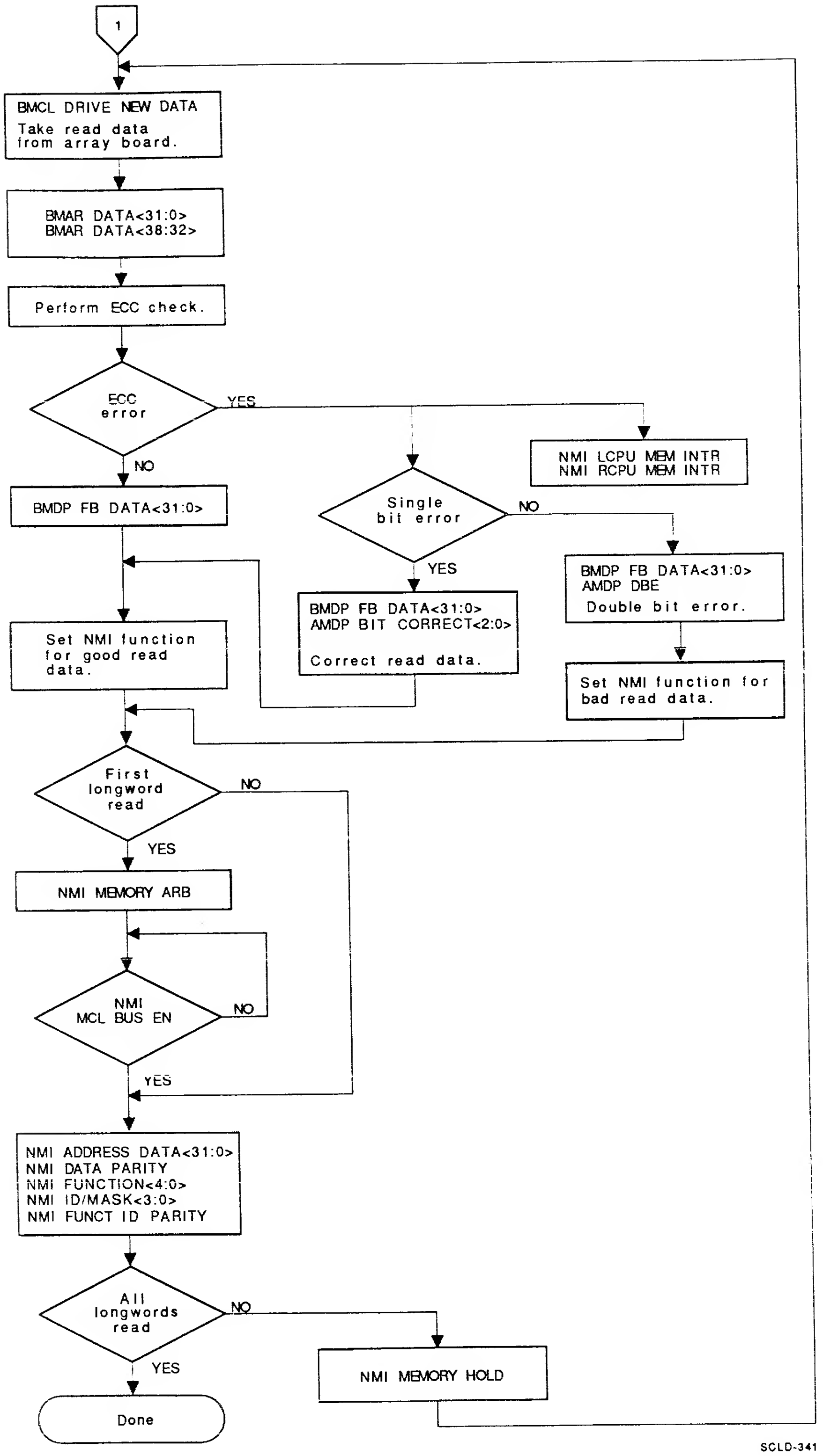


Figure 1-6 Read Data Cycle Flow Diagram (Sheet 2 of 2)

memory, it asserts NMI MCL BUS EN. NMI MCL BUS EN enables the DFA output lines onto the NMI. The DFA outputs are:

- The read data (NMI ADDRESS DATA<31:0>).
- A data parity bit generated in the DFA (NMI DATA PARITY).
- The function code specifying the data as good "read-return" data.
- The ID of the commander that requested the data (NMI ID MASK<3:0>). (The ID was stored during the command/address sequence, Section 1.4.5.1.)
- A function/ID parity bit generated in the DFA (NMI FUNCT ID PARITY).

If the function taken off the NMI was a longword read, the operation is finished. If the function taken off the NMI is an octaword read, the DFA asserts NMI MEMORY HOLD to hold the NMI. This eliminates the need of re-arbitrating for the NMI for the second, third, and fourth longwords of read data.

The MRM holds the array board read-enabled by keeping AMCL READ BD SEL asserted, and then re-asserts BMCL DRIVE NEW DATA to take the next longword of read data along with its ECC check bits. The process repeats until all the longwords are read. The NMI function code generated for the second, third, and fourth longwords is for good "read/continue" data.

If a single-bit ECC error is detected during an ECC check, the MDP generates correction bits (AMDP BIT CORRECT<2:0>) which are sent to the DFA along with the feedback read data (BMDP FB DATA<31:0>). The data is corrected in the DFA and then outputs onto the NMI as good read data.

If a double-bit ECC error is detected, the MDP outputs AMDP DBE to the DFA along with the feedback read data. The DFA generates a "bad read data" function code which is placed on the NMI along with the erroneous read data and the other NMI signals. The bad-data function code may be a read/return or a read/continue function code, as the case may be.

The DFA, sensing the MDP correction bits or the double-bit error signal from the MDP, asserts a CPU interrupt on the NMI (NMI LCPU MEM INTR and NMI RCPU MEM INTR) to indicate that an ECC error has been detected.

1.4.5.4 Masked Write

-- In a masked write opeation, write data is placed into the MDP during the write-data cycle(s) as described in Section 1.4.5.2. Data is then read from the addressed location on the selected array board and used to overwrite the data stored in the MDP. The new data is then written back into the array board.

Figure 1-7 is a flow diagram of a masked-write operation and is used in conjunction with the block diagram (Figure 1-3).

In a masked write operation data cycle, the commander places the following signals on the NMI:

- NMI FUNCTION<4:0> -- the command function (write data)
- NMI ID MASK<3:0> -- mask field
- NMI FUNCT ID PARITY -- parity over the function and ID/mask fields
- NMI ADDRESS DATA<31:00> -- the write data longword
- NMI DATA PARITY -- parity bit for the write data

NMI FUNCTION<4:0> specifies a write-data function to the MCL.

Using the data parity bit and the function/ID/mask parity bit from the NMI, the DFA performs parity checks on the write data and the function/ID/mask fields. If an error is detected, the DFA asserts NMI FAULT on the NMI and ADFA PARITY ERROR1 to the MDP. ADFA PARITY ERROR1 causes the MDP to set the bad-data bit (AMCL BAD DATA) on the NAB. The write operation is allowed to execute, however, when the write data is written into the arrays, the asserted bad-data bit marks the data as bad. The data is written so that it can be analyzed later by maintenance diagnostics.

The write data (NMI ADDRESS DATA<31:0>) passes through the DFA to the MDP as ADFA DATA ADDR<31:0>. The write data is stored in the MDP by AMRM INPUT WRT EN<3:0> from the MRM.

NMI ID MASK<3:0> is transferred to the MRM as ADFA MASK<3:0> where it is stored for use in overwriting the write data in the MDB.

If this is a multi-longword masked write operation, the process repeats until all the longwords are stored in the MDB and their associated mask fields are stored in the MRM.

The MRM then issues a read command to the selected array board which asserts its BMAR SEND NO CMD, and then BMAR DATA RDY DONE when the arrays have been read. The MRM then asserts the read-board select line and the drive-new data line to transfer the read data and check bits to the MDP where an ECC check is made on the data.

If no ECC error is found, the MDP uses AMRM FB WRT EN<3:0> from the MRM to input the read data into the MDB, thereby overwriting specific bytes of the stored write data. AMRM FB WRT EN<3:0> is derived from the mask field(s) stored in the MRM. Hence, the read data overwrites write data according to the mask field.

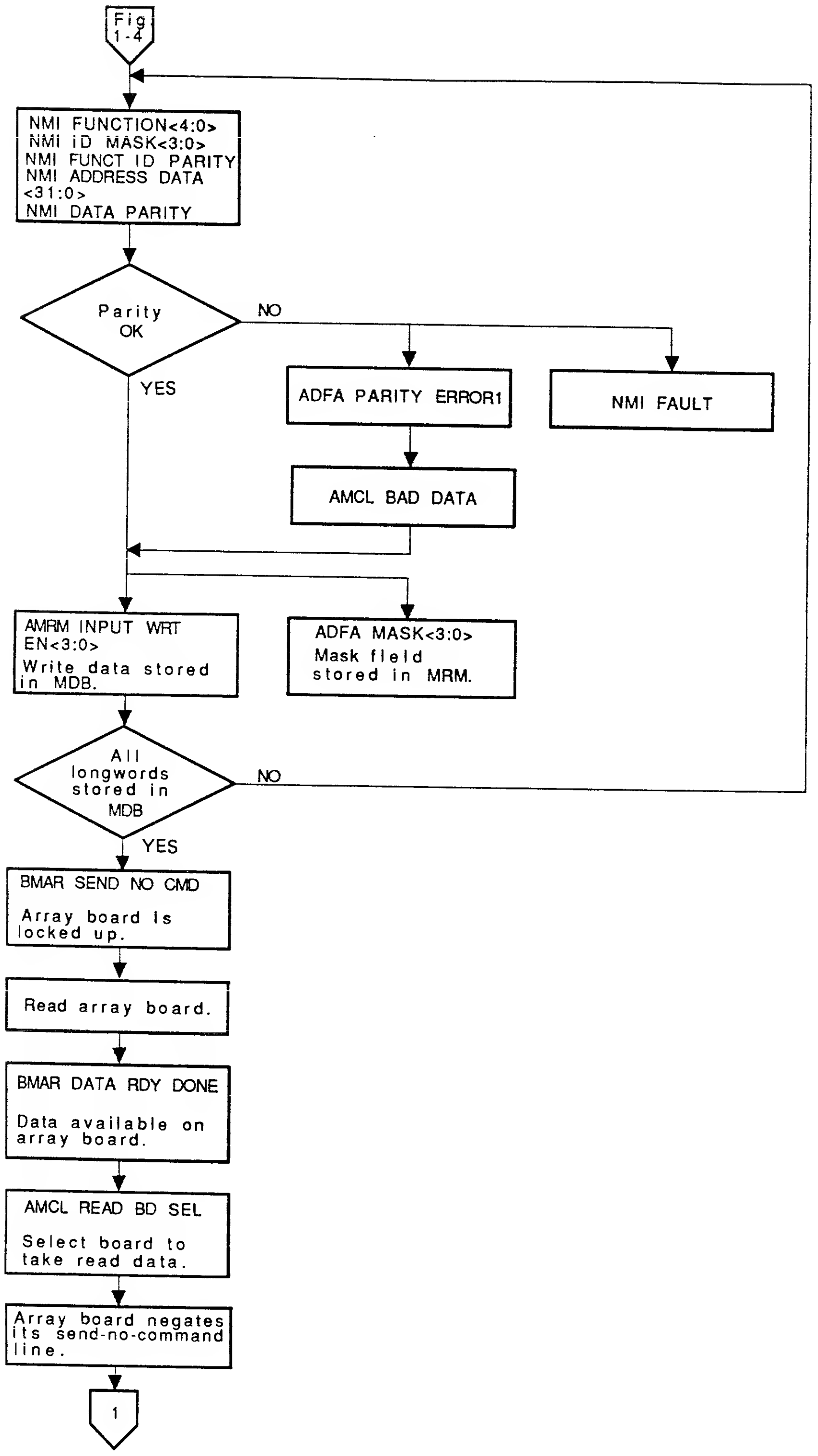


Figure 1-7 Masked Write Data Cycle Flow Diagram (Sheet 1 of 2)

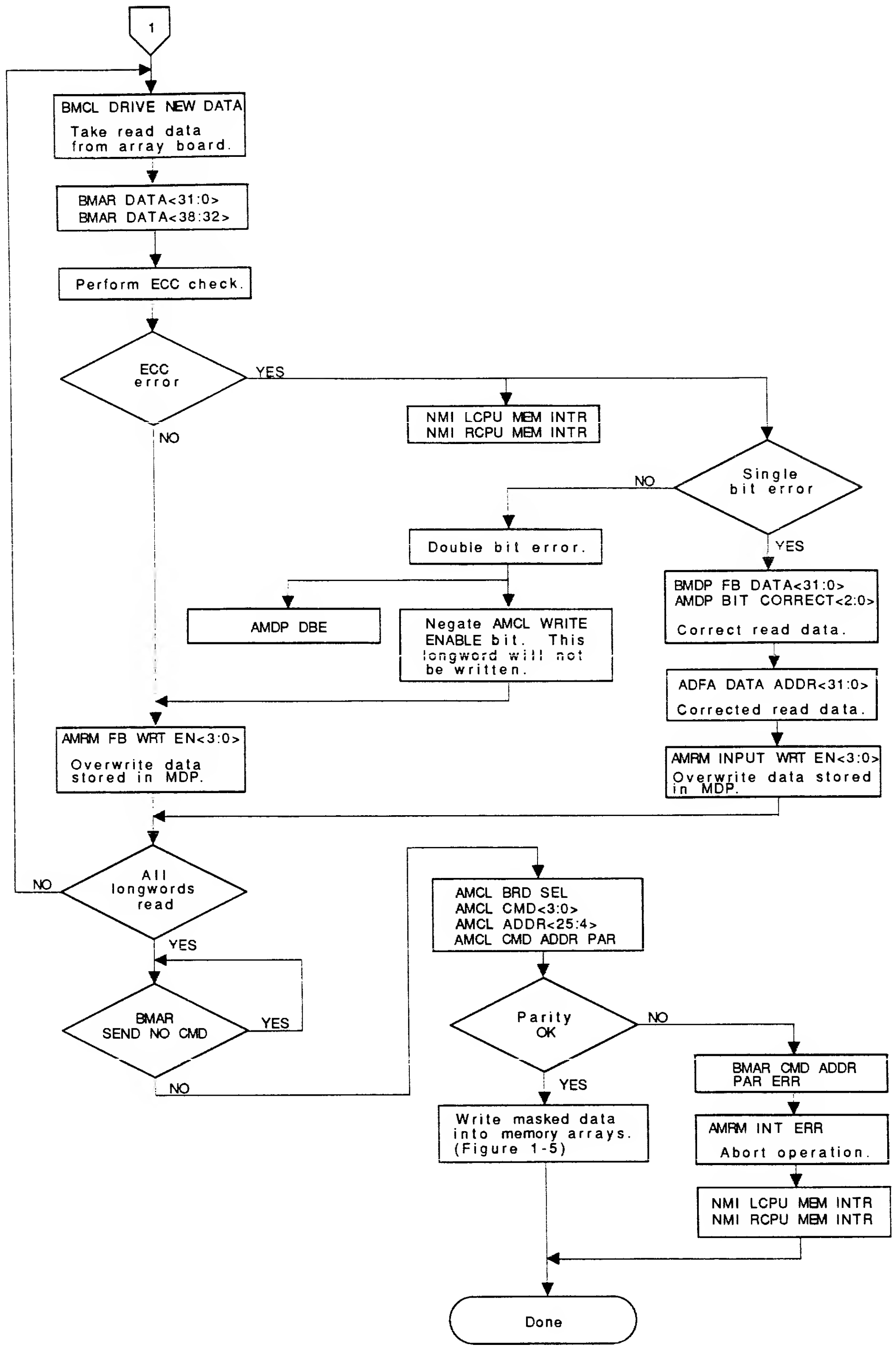


Figure 1-7 Masked Write Data Cycle Flow Diagram (Sheet 2 of 2)

If this is a multiword operation, the process repeats until all of the read data is taken from the array board and entered into the MDP.

The MRM then transfers the new data from the MDP to the array board with a write command. The BMAR SEND NO CMD line is checked and if false, the MRM asserts the board's select line, the write-longword command, and the command/address parity bit. The MDP outputs the address to the array board where a parity check is made. If the parity checks OK, the masked data is written into the arrays as described in Section 1.4.5.2.

If a single-bit error is found in a longword of data read from the array board, the MDP routes the longword (BMDP FB DATA<31:0>) to the DFA, along with error correction bits (AMDP BIT CORRECT<2:0>). The data longword is corrected in the DFA and then wrapped back to the MDP, via the ADFA DATA ADDR<31:0> path, where it overwrites specific bytes of the stored data via enabling bits AMRM INPUT WRT EN<3:0>. The MRM senses that the read data had to be corrected in the DFA and is inputing into the MDP via the ADFA DATA ADDR<31:0> path from the DFA. Consequently, it applies the associated mask code to the AMRM INPUT WRT EN<3:0> enabling signals, thus overwriting the write data according to the mask field.

If a double-bit error is found in a longword of data read from the array board, the MDP negates the write-enable bit (AMCL WRITE ENABLE) associated with the longword. The MRM then proceeds to overwrite the data stored in the MDP with the bad read data, using the AMRM FB WRT EN<3:0> mask code, just as if there were no error. When the data is written into the array module, the longword(s) containing a double-bit error will have its write-enabled bits negated and will not be written into the array bank(s). This preserves the original erroneous data in the array for analysis by maintenance routines.

When an ECC error occurs, the DFA, sensing the correction bits or the AMDP DBE double-bit error signal, asserts the CPU interrupt on the NMI to indicate that an ECC error has been detected.

1.4.6 Four-Megabyte Memory Array Board (MAR4)

The MAR4s share common signal lines on the NAB except for the following four signals:

- AMCL BRD SEL
- AMCL READ BD SEL
- BMAR SEND NO CMD
- BMAR DATA RDY DONE

There are eight lines on the NAB for each of the above signals with one line connected to each MAR4 slot on the backplane.

The NAB carries board select signals, command/address signals, and data between the MCL and the MAR4s. Separate data lines exist on the NAB for read data and for write data. Thirty-two write data lines carry write data from the MCL to the MAR4. Thirty-nine* read data lines carry read data from the MAR4 to the MCL.

The MAR4 board has a total storage capacity of four megabytes. The board has four memory array banks with each bank containing one megabyte of memory. The array banks have a 39-bit wide common I/O to the MAR4's internal data bus. The 39 bits consists of a 32-bit longword and seven ECC check bits.

The MCL performs three command opertions on a MAR4 board. These are:

- Longword write -- Writes a data longword and seven ECC check bits into one of the MAR4's array banks.

- Longword read -- Reads a data longword and seven ECC check bits from one of the MAR4's array banks.

- Octaword read -- Reads four data longwords and their associated check bits from a MAR4 board. The data is read from all four array banks (one longword from each bank). The first longword transferred to the MCL can be from any bank. The other three longwords are transferred from the following three banks in wrap-around order. For example, if the longword from bank 2 is the first longword transferred, the next longwords transferred are from banks 3, 4, and 1, in that order.

In addition, the MCL provides a refresh clock to the MAR4 boards which is used by MAR4 refresh logic to refresh the arrays on a periodic basis. Battery backup is provided to maintain array refreshes during periods of power interruptions.

Figure 1-8 is a simplified flow diagram of MAR4 write and read operations. Refer to it along with the block diagram (Figure 1-3) in the following discussion.

1.4.6.1 MAR4 Select and Command/Address -- The MCL initiates an operation by asserting a board-select signal, the command/address, and the C/A parity bit on the NAB.

The MCL selects the MAR4 by asserting its AMCL BRD SEL line. AMCL BRD SEL selects the MAR4 by enabling the MAR4 control logic.

AMCL CMD<3:0> is a four-bit command applied to the MAR control logic. Figure 1-9 illustrates the command fields. AMCL CMD<3:2> are function bits specifying the operation that is to be performed. AMCL CMD<3> is the write bit which is negated for a

* A 32-bit longword and 7 ECC check bits.

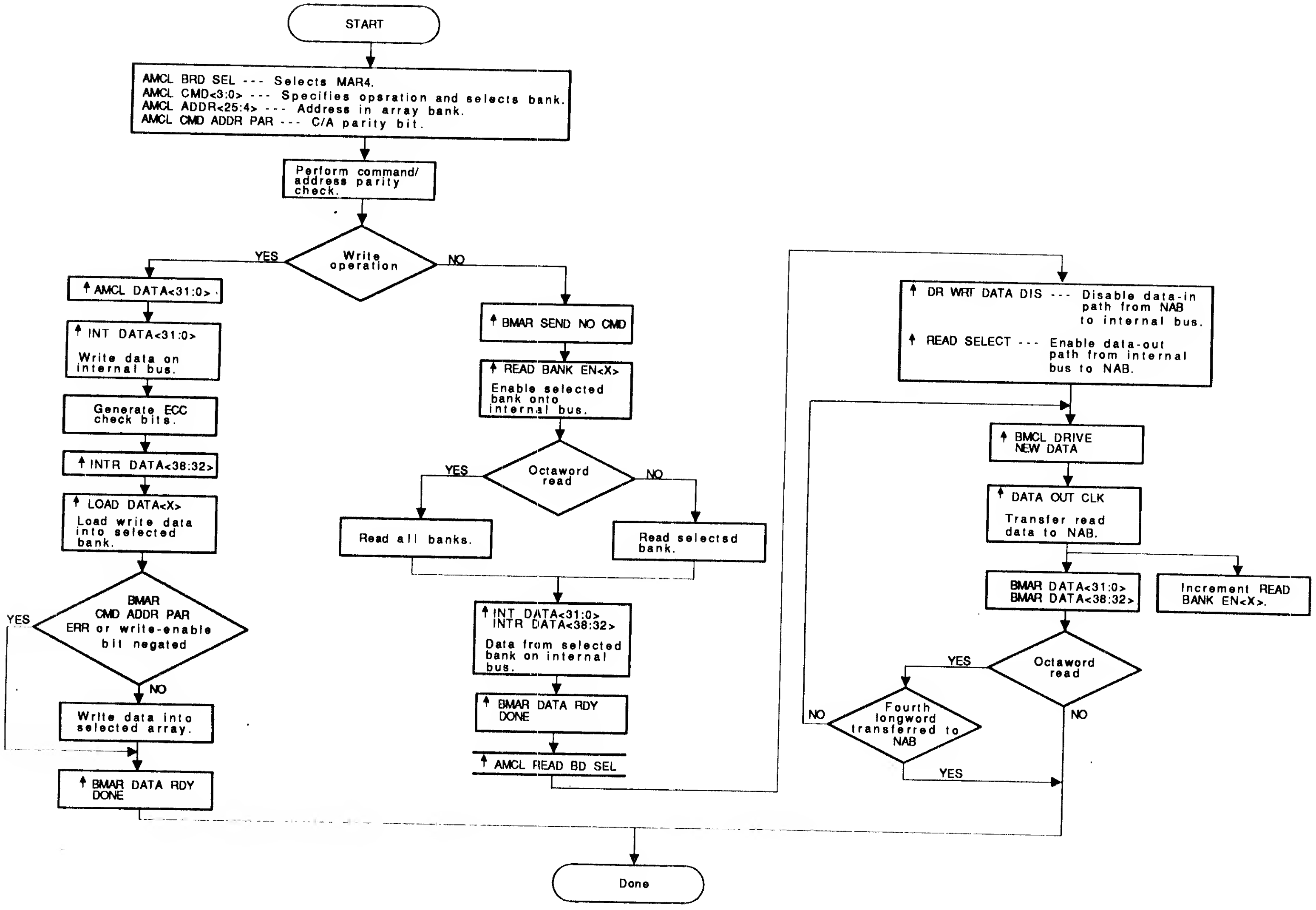

START
AMCL BRD SEL --- Selects MAR4.
AMCL CMD<3:0> --- Specifies operation and selects bank.
AMCL ADDR<25:4> --- Address in array bank.
AMCL CMD ADDR PAR --- C/A parity bit.
Perform command/ address parity check.
Write operation
YES
NO
AMCL DATA<31:0>
INT DATA<31:0> Write data on internal bus.
Generate ECC check bits.
INTR DATA<38:32>
LOAD DATA<X> Load write data into selected bank.
BMAR CMD ADDR PAR ERR or write-enable bit negated
YES
NO
Write data into selected array.
BMAR DATA RDY DONE
BMAR SEND NO CMD
READ BANK EN<X> Enable selected bank onto internal bus.
Octaword read
YES
NO
Read all banks.
Read selected bank.
INT DATA<31:0> INTR DATA<38:32> Data from selected bank on internal bus.
BMAR DATA RDY DONE
AMCL READ BD SEL
DR WRT DATA DIS --- Disable data-in path from NAB to internal bus.
READ SELECT --- Enable data-out path from internal bus to NAB.
BMCL DRIVE NEW DATA
DATA OUT CLK Transfer read data to NAB.
BMAR DATA<31:0> BMAR DATA<38:32>
Increment READ BANK EN<X>.
Octaword read
YES
NO
Fourth longword transferred to NAB
NO
YES
Done
SCLD-344

Figure 1-8 MAR4 Read/Write Flow Diagram

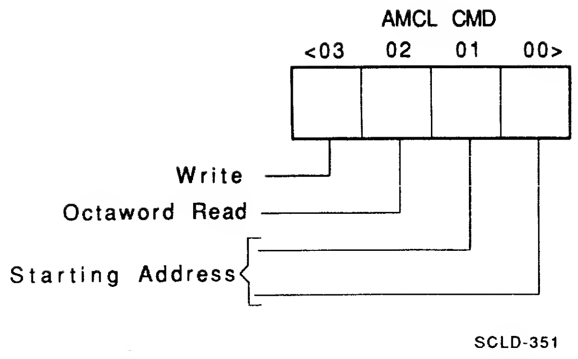


Figure 1-9 MAR4 Command Fields

write and asserted for a read*. AMCL CMD<2> is the octaword read bit which is asserted for an octaword read and negated for a longword read. AMCL CMD<1:0> is the starting address which specifies the bank to be accessed for a longword read or write. For an octaword read, it specifies the first bank to transfer its data to the MCL.

AMCL ADDR<25:4> is the array address to be accessed.

AMCL CMD ADDR PAR is a parity bit on the command/address.

The command/address and the command/address parity bit are applied to a command/address parity checker. If a parity error is detected, BMAR CMD ADDR PAR ERR is asserted back to the MCL which aborts the operation and asserts the memory-interrupt lines on the NMI.

1.4.6.2 **Write Operation** -- If a write operation is executing, the MCL places the write data (AMCL DATA<31:0>) on the NAB. The write data is coupled onto the MAR4's internal data bus as INT DATA<31:0>. The data is applied to all of the array banks and to an ECC generator. The ECC generator generates 7 ECC check bits on the data and applies them to the banks as INTR DATA<38:32>.

AMCL ADDR<25:4> is applied to all four banks.

The MAR4 control logic responds to the input write command by asserting LOAD DATA<X> where X is the number of the bank selected by the command starting address. LOAD DATA<X> loads the write data (longword and check bits) into the selected bank but does not write the data into the array.

* The write bit is inverted on the MAR4 board where its asserted state specifies a write command.

After the write data is loaded into the bank, it is written into the array providing that there was no command/address parity error and the longword's write-enable bit is set. If a command/address parity error did occur, or the write-enable bit is not set (negated), no data is written.

A write-data parity error detected in the array board will not prevent the data from being written but it will mark it as bad data.

The control logic then asserts BMAR DATA RDY DONE on the NAB to inform the MCL that the write operation has been completed and it can issue a new command to the array bank. BMAR DATA RDY DONE is asserted whether or not the write data was written.

1.4.6.3 Read Operation -- If a read operation is executing, the control logic asserts the MAR4's BMAR SEND NO CMD line on the NAB to inhibit the MCL from issuing any new commands to the MAR4 until the read operation is complete. A read operation executes in two segments; a read of the array(s) and a transfer of the read data to the MCL. The two segments do not necessarily follow one another immediately. After the array(s) have been read the MCL will take the read data when it is ready. BMAR SEND NO CMD serves as a "read lock" signal between the two segments. It asserts during the "read array" segment and inhibits the MCL from issuing any new commands to the MAR4* until the "data transfer" segment is started.

The MAR4 control logic responds to the input read command by asserting READ BANK EN<X> where X is the number of the bank selected by the command starting address. READ BANK EN<X> enables the output of the selected bank onto the MAR4 internal bus.

The array banks(s) are then read according to the type of read in progress. If a longword read is executing, the selected bank is read and the data longword and associated check bits are coupled to the internal bus as INT DATA<31:0> and INTR DATA<38:32> respectively. If an octaword read is executing, all of the banks are read and the data from the selected bank is coupled to the internal bus. The data read from the other three array banks is held within the banks.

The control logic then asserts BMAR DATA RDY DONE on the NAB to inform the MCL that the "read array" segment has been completed and it can start the "data transfer" segment when it is ready.

Note that a command/address parity error has no effect on a read operation within the MAR4.

* The MCL may issue commands to other MAR4 boards on the NAB.

The MCL initiates the "data transfer" segment by asserting the MAR4's read board-select line (AMCL READ BD SEL). The MAR4 control logic responds to AMCL READ BD SEL by asserting DR WRT DATA DIS and READ SELECT. DR WRT DATA DIS disables the data-in path from the NAB to the MAR4's internal bus. READ SELECT enables the data-out path from the internal bus to the NAB.

The MCL then asserts BMCL DRIVE NEW DATA to the control logic which in turn asserts DATA OUT CLK. DATA OUT CLK transfers the read data from the internal bus to the NAB via flip-flops. The data longword and associated check bits appear on the NAB as BMAR DATA<31:0> and BMAR DATA<38:32> respectively.

In addition, DATA OUT CLK increments READ BANK EN<X> within the control logic to enable the read data (if any) from the next bank out onto the internal bus. This function is used for octaword read operations.

If a longword read operation is being executed, the operation is complete.

If an octaword read operation is being executed, the MCL re-asserts BMCL DRIVE NEW DATA to transfer the second longword (and its check bits) out to the NAB. DATA OUT CLK again increments READ BANK EN<X> to couple the third longword (not necessarily from the third bank) to the internal bus.

BMCL DRIVE NEW DATA continues to assert causing READ BANK EN<X> to step through all four array banks taking a data longword and its associated check bits from each bank.

When all four banks have been read, the octaword read operation is complete.

CHAPTER 2
MEMORY CONTROLLER (MCL)

The MCL (memory controller) is divided into three areas as discussed in Chapter 1. The three areas are:

- The DFA (DAD, FUNK, and ARID MCAs)
- The MDP (memory data path)
- The MRM (MSC, MSC1, RSC, and MASC MCAs)

Overviews of the DFA, MDP, and MRM are given in Sections 2.1, 2.5, and 2.9 respectively. These three areas can be read as a general description of the MCL. Each overview is followed by a detailed description of the MCAs used in the respective area.

2.1 DFA OVERVIEW (Figure 2-1)

The three MCAs that make up the DFA are defined below.

- DAD = data/address: Interfaces the data and address to the NMI.

- FUNK = function: Interfaces the function field to the NMI, decodes the function field, generates the NMI confirmation code, and generates NMI faults.

- ARID = arbitration/ID: Processes data and function parity, processes the mask and commander ID fields, arbitrates for the NMI, and generates NMI interrupts and busy requests.

2.1.1 Command/Address Cycle

In the command/address cycle, the following is placed on the NMI by the NMI commander.

- NMI FUNCTION<4:0> -- command function

- ID MASK<3:0> -- commander's ID

- FUNCT ID PARITY -- parity bit for function field and commander ID

- ADDRESS DATA<31:0> -- memory address
- NMI DATA PARITY -- address parity bit

The five-bit function field (NMI FUNCTION<4:0>) is applied to a parity generator in the FUNK where parity is generated for the function field. The generated parity bit (CTRL GEN PARITY) is applied to a parity checker in the ARID.

The function field specifies the commanded function as shown in Table 1-2. The function field is decoded in the FUNK function logic, which outputs a three-bit command code (BDFA CMD<2:0) and a two-bit size field (BDFA SIZE<1:0>). The three-bit command code specifies a read or write operation, a memory or CSR access, and a non-masked or masked operation, as shown in Table 2-1.

Table 2-1 Command Code

BDFA CMD <2 1 0>	Command
0 0 0	Read memory
0 0 1	No Op
0 1 0	Read CSR
0 1 1	No Op
1 0 0	Write memory
1 0 1	Write memory masked
1 1 0	Write CSR
1 1 1	No Op

```
 ¶  ¶  ¶
 ¶  ¶   -------Non-masked/masked
 ¶     --------Memory/CSR
     ----------Read/write
```

The function logic uses address bit 29 (NMI ADDRESS DATA<29>) to determine if the command function is to memory or to a CSR.

The two-bit size code specifies the size of the operation as shown in Table 2-2.

Table 2-2 Size Code

BDFA SIZE <1 0>	Size
0 0	Longword
0 1	Octaword
1 0	Quadword
1 1	Hexword

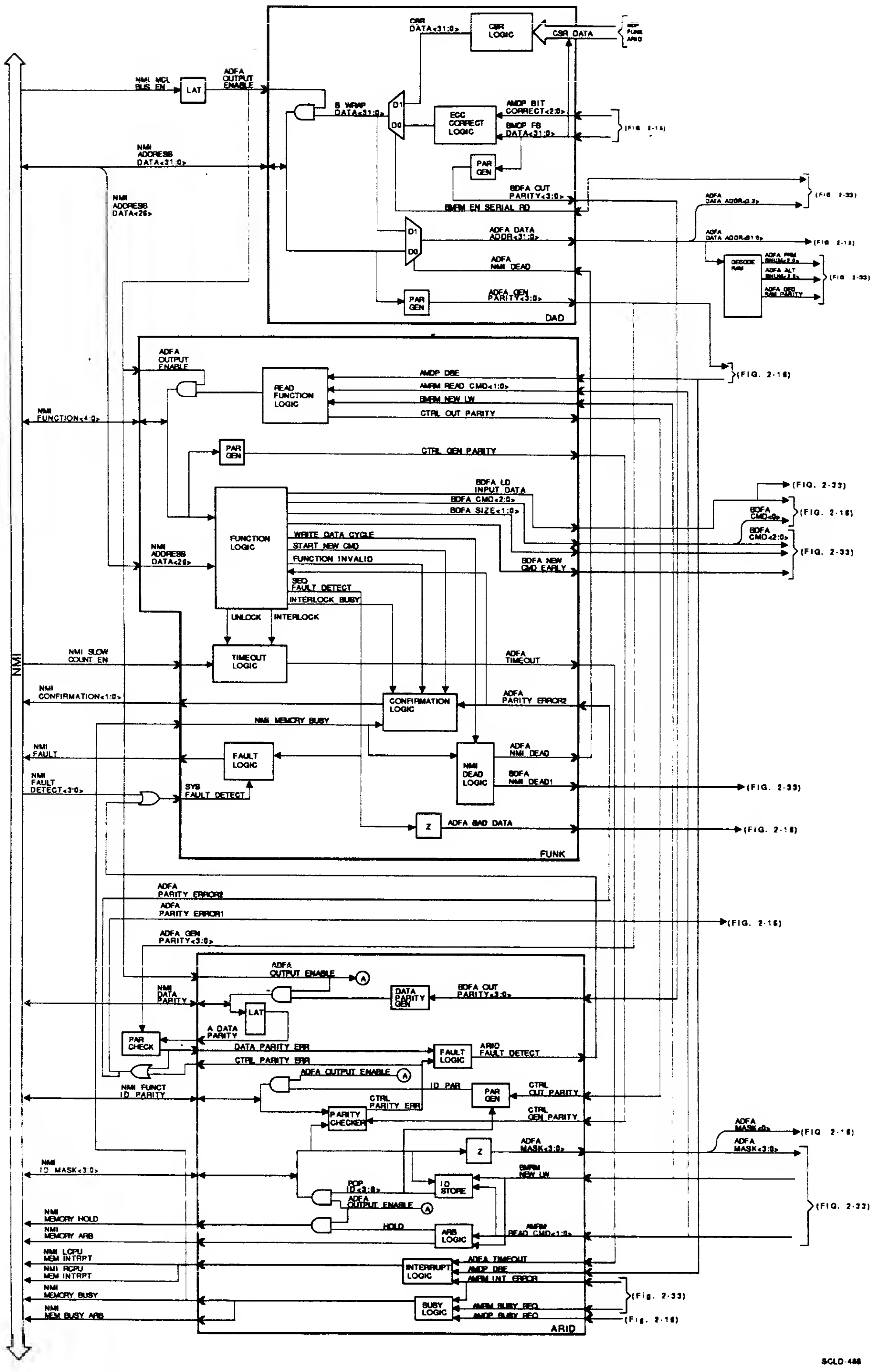


Figure 2-1 DFA Block Diagram

The command code and size field are sent to the MRM for execution of the command function.

If the function logic detects an invalid function code, it aborts the operation and asserts FUNCTION INVALID to the confirmation logic which will assert a "no acknowledgement" confirmation code (Table 1-4) on the NMI during the confirmation cycle (the confirmation cycle is the second cycle after the command/address cycle; see Table 1-3 and Tables 1-5 through 1-9). If a valid function is decoded and no parity errors are detected (ADFA PARITY ERROR2 false), START NEW CMD is asserted to the confirmation logic causing it to place a "command accepted" code on the NMI during the confirmation cycle.

If a read interlock command is decoded, INTERLOCK is asserted to the timeout logic which then starts counting NMI SLOW COUNT EN clocks from the NMI. When a write unlock command is received, the function logic asserts UNLOCK which resets the timeout logic. If a write unlock command is not received within approximately thirteen microseconds from a read interlock command, the timeout logic asserts ADFA TIMEOUT to the interrupt logic in the ARID causing it to assert an interrupt on the NMI.

If a read interlock command is received while a previous read interlock command is pending (previous read interlock command received but a write unlock command has not been received yet), the function logic aborts the new read interlock command and asserts INTERLOCK BUSY to the confirmation logic. The confirmation logic places an "interlock busy" confirmation code on the NMI during the confirmation cycle.

Another output of the FUNK function logic is BDFA NEW CMD EARLY which is used to inform the MRM that a new command has been received.

The commander ID (NMI ID MASK<3:0>) is applied to the ARID where it is stored in the ID store logic. When a read command is executed, the commander ID will be popped from the ID store logic and placed on the NMI along with the read data.

The commander ID is also applied to the ARID parity checker where parity is generated on the ID and combined with the function parity bit from the FUNK (CTRL GEN PARITY). The composite parity bit is compared to the function-ID parity bit from the NMI (NMI FUNCT ID PARITY) and if an error is detected, CTRL PARITY ERR is asserted to the ARID fault logic and to an OR function outside the ARID. The ARID fault logic asserts ARID FAULT DETECT which causes SYS FAULT DETECT to assert to the fault logic in the FUNK. The FUNK fault logic then asserts NMI FAULT on the NMI.

When the OR function outside the ARID receives CTRL PARITY ERR, it asserts ADFA PARITY ERROR2 to the FUNK function logic causing it to abort the operation. ADFA PARITY ERROR2 is also applied to the FUNK confirmation logic which will place a "no acknowledgement" confirmation code on the NMI during the confirmation cycle.

The memory address (NMI ADDRESS DATA<31:0>) is applied to the DAD where it passes through a mux and outputs the DAD as ADFA DATA ADDR<31:0>. ADFA DATA ADDR<31:0> is sent to the MDP where it is stored in the MDB (memory data buffer). Data-address bits <3:2> are sent to the MRM where they select the array bank to be accessed.

ADFA DATA ADDR<31:0> is also applied to a decode RAM where it selects the primary and alternate array board for the command operation. ADFA PRM BNUM<2:0> is the primary board number and is used to select the array board for all write operations, for longword and octaword read operations, and for the first octaword transfer of a hexword read operation. ADFA ALT BNUM<2:0> is the alternate board number and is used only for the second octaword transfer of a hexword read operation. The primary and alternate board numbers are sent to the MRM where they are used to select the array board(s) to be accessed. A decode-RAM parity bit (ADFA DEC RAM PARITY) is also output from the decode RAM for use in an MRM parity check.

Parity is generated on the memory address and outputs the DAD as ADFA GEN PARITY<3:0> where it is sent to the MDP for storage in the memory data buffer. In addition, ADFA GEN PARITY<3:0> is sent to a parity checker where it is checked against the address parity bit (NMI DATA PARITY) from the NMI. The address parity bit is latched in the ARID and then output to the parity checker as A DATA PARITY. If the parity checker detects an address parity error, it asserts DATA PARITY ERR to the ARID fault logic which asserts ARID FAULT DETECT. This results in the assertion of NMI FAULT on the NMI as in the case of a function/ID parity error.

In addition, DATA PARITY ERR asserts ADFA PARITY ERROR2 to the FUNK function logic and confirmation logic, resulting in the operation being aborted and a "no acknowledgement" confirmation code for the NMI, as in the case of a function/ID parity error.

2.1.2 Write Data Cycle(s)

Write-data cycle(s) follow the command/address cycle for all write operations. For longword operations, only one write-data cycle occurs. For multi-longword operations, more than one write-data cycle occurs (see Tables 1-3, 1-5, and 1-6).

In a write-data cycle, the following is placed on the NMI by the NMI commander.

- NMI FUNCTION<4:0> -- command function
- NMI ID MASK<3:0> -- byte mask
- NMI FUNCT ID PARITY -- parity bit for function field and byte mask

- NMI ADDRESS DATA<31:0> -- write data
- NMI DATA PARITY -- write data parity bit

A function parity bit (CTRL GEN PARITY) is generated for the function field as was done during the command/address cycle.

For each write-data cycle, the FUNK function logic outputs BDFA LD INPUT DATA to the MDP to load the associated write data into the memory data buffer, and to the MRM to load the mask field into mask store logic. It also outputs WRITE DATA CYCLE to the NMI dead logic for use in single-bit error correction during masked write operations (see Section 2.1.7).

The byte mask (NMI ID MASK<3:0>) is applied to the ARID and then output to the MRM as ADFA MASK<3:0> where it is used for byte selection in masked write operations.

Parity is checked on the byte mask and function field just as in the command/address cycle. The function parity bit (CTRL GEN PARITY) is sent from the FUNK to the parity checker in the ARID where it is combined with the parity bit generated from the byte mask. The composite parity bit is compared to the function and byte mask parity bit and if a parity error is detected, CTRL PARITY ERROR asserts to the fault logic in the ARID and to the OR function outside the ARID. The fault logic causes an interrupt on the NMI. The OR function asserts ADFA PARITY ERROR1 to the MDP where it causes the bad-data bit to assert. Note that the write operation is allowed to continue as opposed to the command/address cycle where a function/commander ID parity error aborted the operation.

The write data (NMI ADDRESS DATA<31:0>) is routed through the DAD (via a mux) and sent to the MDP as ADFA DATA ADDR<31:0>. Parity is generated on the write data and sent to the MDP as ADFA GEN PARITY<3:0>. The write data and parity bits are stored in the MDB in the MDP.

As in the case of the command/address cycle, parity bits ADFA GEN PARITY<3:0> are generated on the write data and sent to the parity checker where it is checked against the write-data parity bit NMI DATA PARITY. If the parity checker detects a data parity error, it asserts DATA PARITY ERR which causes a fault on the NMI via the ARID and FUNK fault logic. In addition, DATA PARITY ERR asserts ADFA PARITY ERROR1 to the MDP where it causes the bad-data bit to assert as in the case of a function/mask parity error. Note that the write operation is allowed to continue as opposed to the command/address cycle where a memory address parity error aborted the operation.

If the function logic in the FUNK, detects missing write data (that is, an octaword write operation is executing but the NMI commander sends only three longwords of write data), it asserts SEQ FAULT DETECT. SEQ FAULT DETECT does the following:

- Asserts NMI FAULT on the NMI via the FUNK fault logic.
- Asserts ADFA BAD DATA to the MDP where it causes the bad-data bit to assert.

2.1.3 First Read Data Cycle

When a read operation is executing and read data has been obtained from the arrays, the MCL arbitrates for control of the NMI by asserting NMI MEMORY ARB. NMI MEMORY ARB is generated by the ARID arbitration logic which receives BMRM NEW LW and AMRM READ CMD<1:0> from the MRM. BMRM NEW LW signifies that a longword of read data has been read from the arrays. AMRM READ CMD<1> specifies that the data is true read data -- not data read as part of a masked write operation. AMRM READ CMD<0> specifies the data as the first longword of the read operation.

Due to pipelining (processing sequential longwords of data at the same time), BMRM NEW LW and AMRM READ CMD<1:0> associated with the next longword (if this is a multi-longword operation) will appear at the ARID even though the MCL has not yet won the bus. AMRM READ CMD<0> specifies a "next" longword to the arbitration logic causing it to assert HOLD to hold the NMI for the next bus cycle once the NMI is won.

The ID store logic in the ARID also receives BMRM NEW LW and AMRM READ CMD<1:0> causing it to output the commander ID (POP ID<3:0>) that was stored during the command/address cycle.

POP ID<3:0> is also applied to a parity generator where it contributes to the generation of a parity bit for the function and ID fields.

The FUNK read-function logic generates a read-return function code for the function field associated with the first longword of read data. The logic receives BMRM NEW LW and AMRM READ CMD<1:0> from the MRM and AMDP DBE (asserted if the read data contains a double-bit error) from the MDP, from which it determines:

- That there is a longword of read data.
- The data is for the NMI.
- That it is the first longword of the read operation.
- Whether or not the data contains a double-bit error.

The read function logic generates a parity bit for the function field which it outputs to the ARID as CTRL OUT PARITY. In the ARID, CTRL OUT PARITY is applied to a parity generator where it is combined with the parity bit generated for the commander ID. The composite parity bit is output as ID PAR.

The first longword of read data is received by the DAD from the MDP, as BMDP FB DATA<31:0>. The data is routed through ECC correction logic where it undergoes single-bit error correction if necessary. If a single-bit error had been detected, AMDP BIT CORRECT<2:0> from the MDP will effect the correction in the ECC correction logic. From the correction logic, the read data passes through a mux to become B WRAP DATA<31:0>.

Read data BMDP FB DATA<31:0> is also applied to a parity generator in the DAD where parity bits BDFA OUT PARITY<3:0> are generated and sent to the ARID. In the ARID, the parity bits are consolidated into a single parity bit by a data parity generator.

During the bus-grant cycle (when the MCL is granted use of the NMI), NMI MCL BUS EN is asserted and becomes ADFA OUTPUT ENABLE. The bus-grant cycle is also the first read data cycle (see Tables 1-7, 1-8, and 1-9). The assertion of ADFA OUTPUT ENABLE does the following:

- Enables the first longword of read data onto the NMI by gating B WRAP DATA<31:0> out of the DAD as NMI ADDRESS DATA<31:0>.
- Enables the read data parity bit out of the ARID onto the NMI as NMI DATA PARITY.
- Enables the read data function field out of the FUNK onto the NMI as NMI FUNCTION<4:0>.
- Enables the commander ID onto the NMI by gating POP ID<3:0> out of the ARID as NMI ID MASK<3:0>.
- Enables the function field and commander ID parity bit onto the NMI by gating ID PAR out of the ARID as NMI FUNCT ID PARITY.
- If this is a multi-longword operation, holds the NMI for the next cycle by gating HOLD out of the ARID as NMI MEMORY HOLD.

2.1.4 "Next" Read Data Cycles

On the second and subsequent read data cycles:

- The read longwords pass through the DAD to the NMI as in the first read data cycle.
- Parity is generated on the read data and placed on the NMI as in the first read data cycle.
- The read function logic in the FUNK senses read command bit <0> to output a read-continue function code. It also monitors AMD DBE to specify a good data or bad data function code.

- The ID store senses read command bit <0> to determine the read data cycles to be "next" cycles and outputs the same commander ID onto the NMI as in the first read data cycle. It does not pop a new commander ID until it senses a new "first" longword of read data.

- Parity is generated on the commander ID and function field and placed on the NMI as in the first read data cycle.

- The arbitration logic keeps NMI MEMORY HOLD asserted so long as the MCL needs the NMI for the next bus cycle. NMI HOLD is false during the fourth data bus cycle. If a hexword read is in progress, the MCL must re-arbitrate for the NMI to output the send octaword of read data.

2.1.5 NMI Interrupt

Interrupt logic in the ARID generates an NMI interrupt when:

- ADFA TIMEOUT asserts from the FUNK timeout logic as discussed in Section 2.1.1.

- AMDP DBE asserts from the MDP indicating an uncorrectable error has been detected.

- AMRM INT ERROR asserts from the MRM indicating an internal parity error or power failure.

2.1.6 NMI Memory Busy

Busy logic in the ARID generates NMI MEMORY BUSY when:

- AMDP BUSY REQ asserts from the MDP indicating the MDB is full and cannot accept any more write data. The MDP also asserts AMDP BUSY REQ when a single-bit error is detected (see Section 2.1.7).

- AMRM BUSY REQ asserts from the MRM indicating that the array boards are busy and no more commands can be accepted.

- AMRM INT ERROR asserts from the MRM. This is done to prevent other nexus from sending commands to memory when an internal error exists.

NMI MEMORY BUSY is also applied to the confirmation logic in the FUNK which will place a "memory busy" confirmation code on the NMI during the confirmation cycle of any command received while NMI MEMORY BUSY is true.

2.1.7 Single-Bit Error Correction During a Masked Write

If the MDP detects a single-bit error in data read from the arrays during a masked write operation, the data is sent to the DAD, along with the AMDP BIT CORRECT<2:0> correction bits, for error correction and is then returned to the MDP.

Upon detecting the single-bit error, the MDP asserts AMDP BUSY REQ to the ARID busy logic which places NMI MEMORY BUSY on the NMI. This is done to halt commands to memory so that the DAD data path can be used for the corrected read data. NMI MEMORY BUSY is applied to NMI dead logic in the FUNK, which then asserts ADFA NMI DEAD to the DAD. ADFA NMI DEAD switches the DAD data mux to select the corrected read data (B WRAP DATA<31:0>) for the return to the MDP.

A copy of ADFA NMI DEAD (ADFA NMI DEAD1) is sent to the mask-store logic in the MRM to signify that the read data is coming from the DAD and not from the NAB.

The NMI dead logic does not assert ADFA NMI DEAD (nor ADFA NMI DEAD1) if a write operation is in progress as indicated by the assertion of WRITE DATA CYCLE from the function logic. The write operation is allowed to finish before the DAD data mux breaks the NMI data path through the DAD.

2.1.8 CSR Reads

CSR logic in the DAD collects CSR serial and parallel data from the DAD, MDP, FUNK, and ARID; converts it to parallel data; and supplies it to the NMI during a CSR read operation. The MRM detects the operaton as a CSR read and asserts BMRM EN SERIAL READ to the DAD. BMRM EN SERIAL READ switches a mux to select CSR DATA<31:0> from the CSR logic for the read data path out to the NMI.

2.2 DATA/ADDRESS (DAD) MCAS

There are four identical DAD MCAs that consist of muxes, ports, and drivers which pass data to and from the NMI. Each DAD carries an eight-bit slice of the 32-bit data/address. A two-bit mode field is applied to each DAD to specify its byte position within the longword. This allows the MCAs to be physically identical but slightly different logically. Areas where the DADs are logically different are noted in the block diagram description.

Figure 2-2 is a block diagram of the DAD MCAs with all four DADs represented on the diagram. The four data slices have been combined giving total bit representation of the signals (for example, the NMI ADDRESS DATA bits are the <31:0> longword rather than the <7:0> byte processed by a single DAD MCA).

There are eight muxes used in forming the various data paths through the DAD MCAs. The block diagram keys the muxes to their names as used in the text.

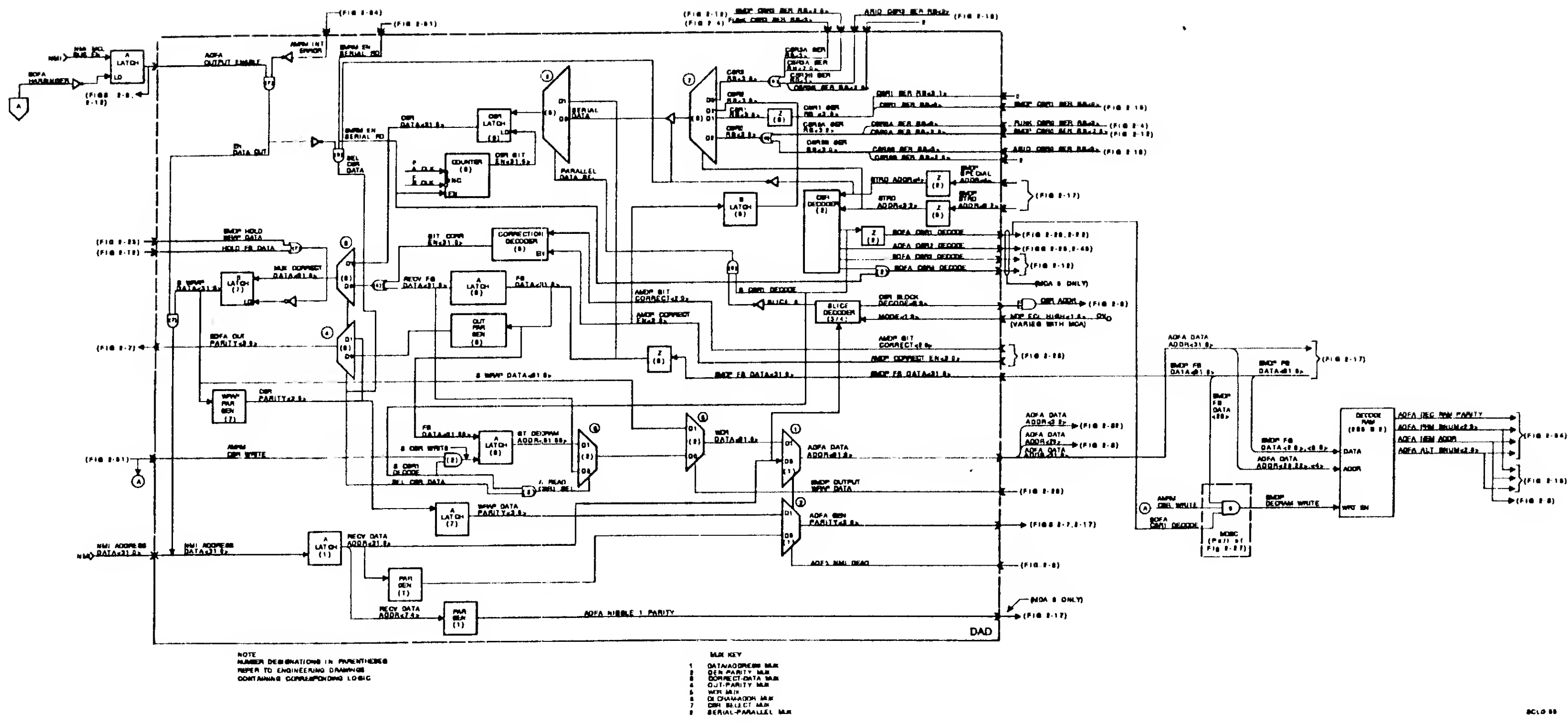
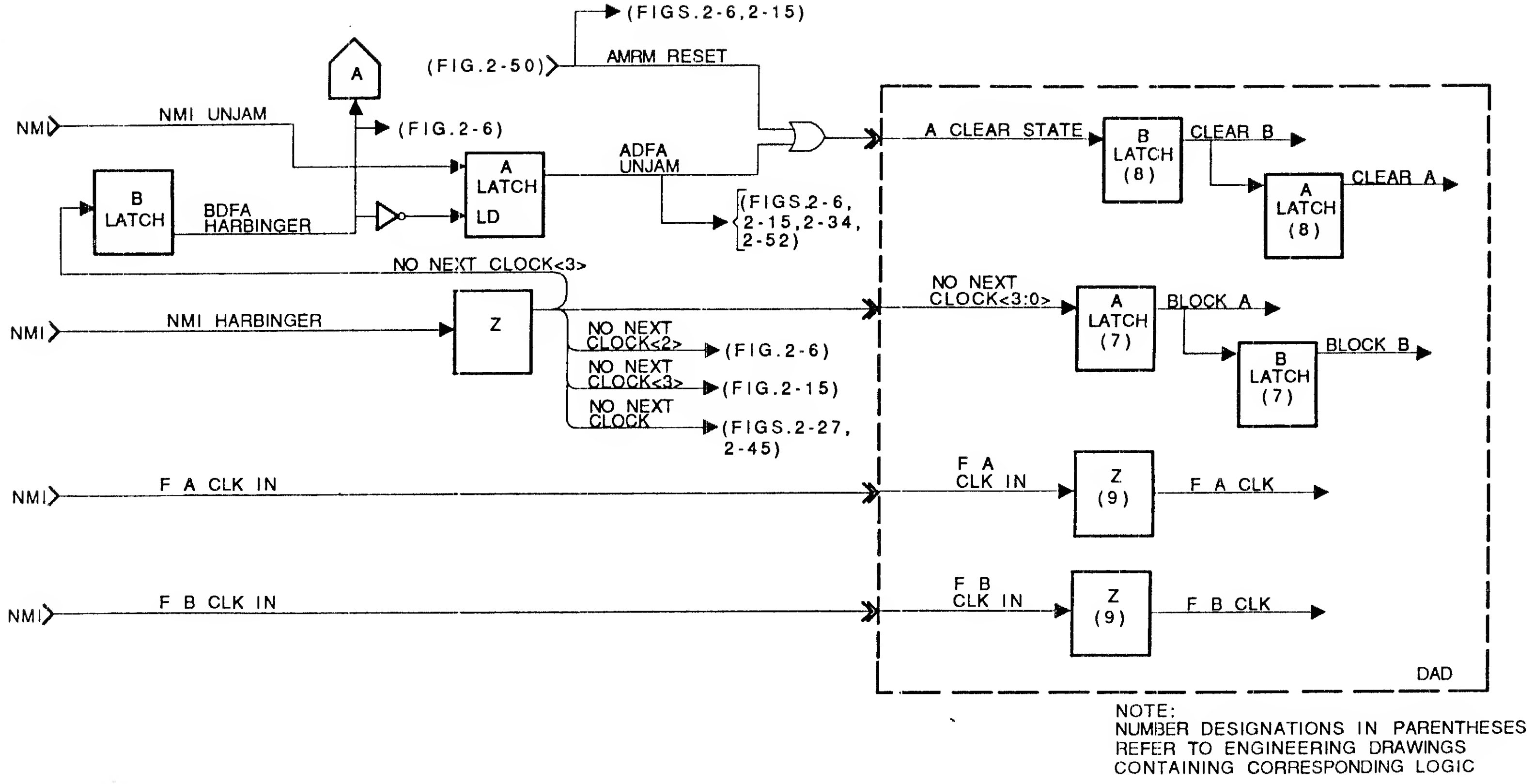


Figure 2-2 DAD Block Diagram (Sheet 1 of 2)



SCLD-40

Figure 2-2 Dad Block Diagram (Sheet 2 of 2)

The major DAD ports are defined and then the block diagram is used to describe how the DAD functions for the various MCL operations.

2.2.1 DAD Ports

The major DAD ports are:

- NMI ADDRESS DATA<31:0> -- data and address I/O to the NMI (bi-directional).
- ADFA DATA ADDR<31:0> -- data-address output port to the MDB.
- BMDP FB DATA<31:0> -- receives feedback data from the MAR4 during NMI reads from memory, masked writes, and decode RAM addressing.
- CSR<3:0> SER RB<3:0> -- input port for serial read back data from CSR0, CSR1, CSR2, and CSR3 to the NMI.

2.2.2 NMI Writes to Memory

Data-address NMI ADDRESS DATA<31:0> from the NMI is latched in the DAD to becomes RECV DATA ADDR<31:0>. RECV DATA ADDR<31:0> is applied to the D0 input of a data-address mux. When ADFA NMI DEAD is false, the data-address is output from the mux as ADFA DATA ADDR<31:0> and transferred to the MDB.

Byte parity is generated on the data-address and is applied to the D0 input of a GEN-parity mux. When ADFA NMI DEAD is false, the mux selects the parity bits and outputs them to the MDB as ADFA GEN PARITY<3:0>. The parity bits are also sent to the ARID where a parity check is made on the NMI data.

Parity is also generated on the four-bit nibble RECV DATA ADDR<7:4>. The generated parity bit (ADFA NIBBLE 1 PARITY) from MCA0 is sent to the MDB where it is used in the generation of address parity for the memory arrays*. The ADFA NIBBLE 1 PARITY outputs from the other three MCAs are not used.

2.2.3 NMI Reads from Memory

Data read from memory is coupled from the arrays to the DAD for transfer to the NMI. The data is input to the DAD at the feedback port as BMDP FB DATA<31:0>. BMDP FB DATA<31:0> becomes FB DATA<31:0> which is then latched to become RECV FB DATA<31:0>.

RECV FB DATA<31:0> is applied to an ECC XOR gate where single-bit error correction occurs (if needed). AMDP CORRECT EN<3:0> and AMDP BIT CORRECT<2:0> are received from the DCHK and applied to a

* Address bits <3:0> are not used.

correction decoder. If a single-bit error was detected by the DCHK, AMDP CORRECT EN<3:0> enables the appropriate byte in the decoder. AMDP BIT CORRECT<2:0> specifies the erroneous bit within the byte. The decoder output (BIT CORR EN<31:0>) is XORed with the memory read data. If no error exists in the data, all 32 bits from the decoder are negated. If a single-bit error exists, the corresponding bit from the decoder is asserted causing the erroneous data bit to be flipped (corrected).

The output from the ECC XOR gate is applied to the D0 input of a correct-data mux which outputs the memory read data as MUX CORRECT DATA<31:0>. MUX CORRECT DATA<31:0> is loaded into a B latch to become B WRAP DATA<31:0>. B WRAP DATA<31:0> is latched by HOLD FB DATA from the ARID arbitration logic, until the ARID has won the NMI bus.

B WRAP DATA<31:0> is gated by EN DATA OUT and then transferred to the NMI via the bi-directional NMI ADDRESS DATA<31:0> port. EN DATA OUT is true when there is no internal error within the MCL (AMRM INT ERROR negated), and ADFA OUTPUT ENABLE is asserted. ADFA OUTPUT ENABLE is asserted by NMI MCL BUS EN which is the NMI bus grant received from the CPU arbitration logic when the MCL has won the bus.

Byte parity is generated on the memory read data by applying the read data (FB DATA<31:0>) to an out-parity generator before it passes through the ECC correction logic. The out-parity generator outputs into the D0 input of an out-parity mux which in turn outputs the parity bits as BDFA OUT PARITY<3:0>. The parity bits are transferred to the ARID MCA where a parity check is made on the memory read data. If single-bit correction occurred on the read data, the ARID compensates for the corrected bit.

2.2.4 Masked Writes Requiring Single-Bit Error Correction

In a masked write operation in which the DCHK detected a single-bit error in the read data, the data is fed into the DAD feedback port, ECC corrected, and then wrapped around and returned to the MDB via the ADFA data-address port. Parity is generated on the wrapped data and returned to the MDB with the data. Note that the wrapped data uses the same output port as the data/address from the NMI. Hence, there must be no input from the NMI while the wrap-data is passing through the DAD. To accomplish this, the ARID asserts NMI BUSY to halt traffic on the NMI whenever the DAD wrap path is to be used. When the FUNK senses that there is no NMI traffic, it asserts ADFA NMI DEAD to enable the wrap-data path. The path of the wrap data through the DAD is described below.

BMDP FB DATA<31:0> is passed through the feedback channel where it undergoes error correction in the ECC XOR gate. The corrected data is then input into the B latch where it becomes B WRAP DATA<31:0>. B WRAP DATA<31:0> is latched by BMDP HOLD WRAP DATA from the MDBC until the MDBC senses that the NMI is quiet (ADFA NMI DEAD true) and negates BMDP HOLD WRAP DATA.

The corrected wrap data held in the B latch (B WRAP DATA<31:0>) is applied to the D1 input of a WOR (wired OR) mux. The MDBC, sensing the true state of ADFA NMI DEAD, asserts BMDP OUTPUT WRAP DATA which causes the WOR mux to select the D1 input for the WOR DATA<31:0> output. WOR DATA<31:0> is then transferred to the ADFA DATA ADDR<31:0> output port via the data-address mux.

The corrected wrapped data (B WRAP DATA<31:0>) is also applied to a wrap-parity generator where parity bits CSR PARITY<3:0> are generated. CSR PARITY<3:0> becomes WRAP DATA PARITY<3:0> which is output from the DAD as ADFA GEN PARITY<3:0> via the GEN-parity mux.

2.2.5 Error-Free Masked Writes

The MAR4 read data is applied to the DAD feedback port even when no error correction is required. This is done for parity checking of the read data. The read data follows the same path as during an NMI read from memory (Section 2.2.3), generating parity bits BDFA OUT PARITY for the ARID where parity is checked. The read data in the DAD never reaches the NMI (ADFA OUTPUT ENABLE false), nor does it wrap back to the MDB (ADFA NMI DEAD false).

2.2.6 Decode RAM Addressing

The wrap data path is also used for addressing of the decode RAM. A DECRAM-ADDR mux outputs store-decode RAM addresses into the D0 input of the WOR mux. The D0 input is selected by the negated state of BMDP OUTPUT WRAP DATA from the MDBC (no single-bit error correction occurring). The DECRAM-ADDR mux inputs are generated when the decode RAM is initially loaded and when reading CSR1.

2.2.6.1 Initially Loading the Decode RAM -- The initial loading addresses for the decode RAM are input at the DAD feedback port (BMDP FB DATA<31:0>). BMDP FB DATA<31:0> becomes FB DATA<31:0> which in turn is latched to become RECV FB DATA<31:0>. RECV FB DATA<31:0> is applied to the D0 input of the DECRAM-ADDR mux. With A READ CSR1 SEL false, the mux selects the D0 input for the WOR mux.

2.2.6.2 Reading CSR1 -- When reading CSR1, the store-decode RAM is addressed at the DAD feedback port (BMDP FB DATA<31:0>). BMDP FB DATA<31:0> becomes FB DATA<31:0> which is applied to the input of an A latch. The latch is enabled for loading by B CSR WRITE obtained by ANDing AMRM CSR WRITE from the MRM and B CSR1 DECODE from a CSR decoder in the DAD.

The CSR decoder receives a 3-bit stored address (BMDP SPECIAL ADDR<4> and BMDP STRD ADDR<3:2>) from the MDB which specifies which CSR is to be accessed. The 3-bit address is decoded and the selected CSR DECODE output is asserted. When the decoder senses that CSR1 is to be accessed, it outputs B CSR1 DECODE.

B CSR WRITE loads the A latch with the store-decode RAM address which appears at the output of the latch as ST DECRAM ADDR<31:0>. ST DECRAM ADDR<31:0> is applied to the D1 input of the DECRAM-ADDR mux where it is selected for the output by the true state of A READ CSR1 SEL.

The A READ CSR1 SEL select signal is obtained by ANDing B CSR1 DECODE with SEL CSR DATA. SEL CSR DATA is asserted by ANDing BMRM EN SERIAL RD (always asserted when CSR0, CSR1, CSR2, or CSR3 is to be read) and the false state of stored address bit <4> (false when CSR0, CSR1, CSR2, or CSR3 is to be accessed).

2.2.7 CSR Reads to the NMI

Four MCL CSRs (CSR0, CSR1, CSR2, CSR3) are read out to the NMI via the DAD serial read-back port. The port receives CSR data, in serial format, from the FUNK, the ARID, and the MDP. The serial data is accumulated and assembled inside the DAD and output to the NMI in parallel format.

The serial data from the four CSRs is applied to a CSR-select mux where two stored address bits from the MDB (STRD ADDR<3:2>) select one of the CSRs. The CSR-select mux outputs the data from the selected CSR to the D0 input of a serial-parallel mux. The path from the CSR-select mux to the serial-parallel mux is enabled by the negated state of STRD ADDR<4>.

The serial-parallel mux selects the D0 input due to the presently false state of PARALLEL DATA SEL. The output from the serial-parallel mux is applied to a 32-bit CSR latch which is load enabled by the CSR BIT EN<31:0> output from a counter. The counter is enabled by EN SERIAL RD and incremented by F A CLK and F B CLK. The counter asserts CSR BIT EN<31:0> in sequence, thereby load enabling the latch bit locations one at a time. As the serial CSR data is applied to the latch it is loaded in a bit at a time.

The serial loading process into the CSR latch is accomplished simultaneously by the four DADs; therefore, each DAD counter outputs eight CSR BIT EN counts in sequence to serially load a byte into each DAD latch. After eight cycles of the counter, the 32-bit CSR latch is fully loaded with the CSR data.

An exception to the serial data format of the received CSR data occurs in the case of CSR1. The least significant byte received (bits <7:0>) is in serial format while the other three bytes are received in parallel format. Consequently, when it is sensed that CSR1 is to be read, the select input to the serial-parallel mux (PARALLEL DATA SEL) is asserted in MCA1, MCA2, and MCA3 to select the D1 input from the feedback port. CSR1 routes its three most significant bytes into the feedback port as FB DATA<31:8>.

PARALLEL DATA SEL in MCA0 is false to select the D0 input from the CSR mux. MCA0 will pass the CSR1 least significant byte through the CSR mux just as in the case for the other CSRs.

PARALLEL DATA SEL is held false in MCA0 by the true state of SLICE 0 from a slice decoder. The slice decoder receives a two-bit code (MODE<1:0>) which specifies the byte position of the MCA according to its location on the MCL module. When the slice decoder senses the MCA to be MCA0, it asserts SLICE 0 which inhibits the AND gate producing PARALLEL DATA SEL. Hence, the serial-parallel mux in MCA0 never selects its D1 input.

The CSR latch outputs the CSR data as CSR DATA<31:0> which is then applied to the D1 input of the correct-data mux. The D1 input to the mux is selected for the output by the true state of SEL CSR DATA. SEL CSR DATA is asserted for a CSR read by the assertion of BMRM EN SERIAL RD and the negation of STRD ADDR<4> and EN DATA OUT.

The CSR data is output from the correct-data mux as MUX CORRECT DATA<31:0> which is loaded into the B latch and then latched by HOLD FB DATA until the MCL has won the NMI. When the MCL receives its bus grant from the CPU, EN DATA OUT asserts and transfers the CSR data to the NMI via the NMI ADDRESS DATA<31:0> port.

The latched CSR data (B WRAP DATA<31:0>) is also applied to the wrap-parity generator where CSR PARITY<3:0> is generated on the CSR data. CSR PARITY<3:0> is applied out to the ARID as BDFA OUT PARITY<3:0> via the D1 input of the out-parity mux.

The slice decoder described in the preceeding discusion, serves another function during a read of the CSRs to the NMI. Address inputs from the NMI are input to the slice decoder from the RECV DATA ADDR<31:0> internal bus. The decoder monitors the addresses and asserts CSR ADDR to the FUNK if a CSR address is detected. The CSR addresses are in the I/O space ranging from 3E000000 for CSR0 to 3E00001C for CSR7.

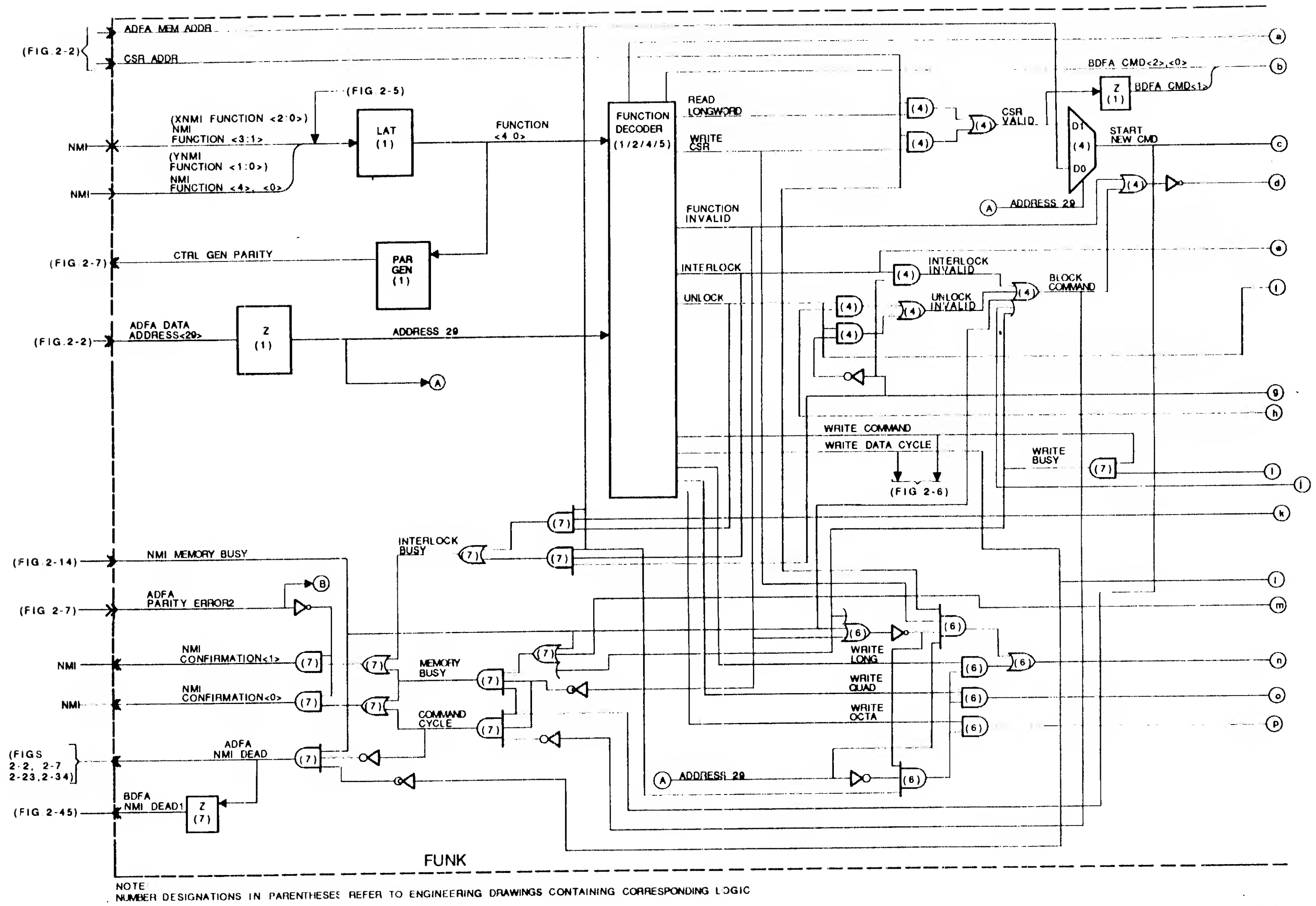
The MODE<1:0> inputs specify which MCA is MCA0, MCA1, etc., so that the decoder in each MCA will monitor the correct byte of the 32-bit address.

2.3 FUNCTION (FUNK) MCA

The FUNK MCA:

- Decodes the NMI function to be used by the MCL.
- Generates read function codes for the NMI.
- Generates the confirmation codes for the NMI.
- Asserts the NMI fault line.
- Provides status bit to CSR0 and CSR1.

Figure 2-3 is a block diagram of the FUNK MCA. Refer to it throughout this section.



NOTE
NUMBER DESIGNATIONS IN PARENTHESES REFER TO ENGINEERING DRAWINGS CONTAINING CORRESPONDING LOGIC

Figure 2-3 FUNK Function and Control Logic (Sheet 1 of 2)

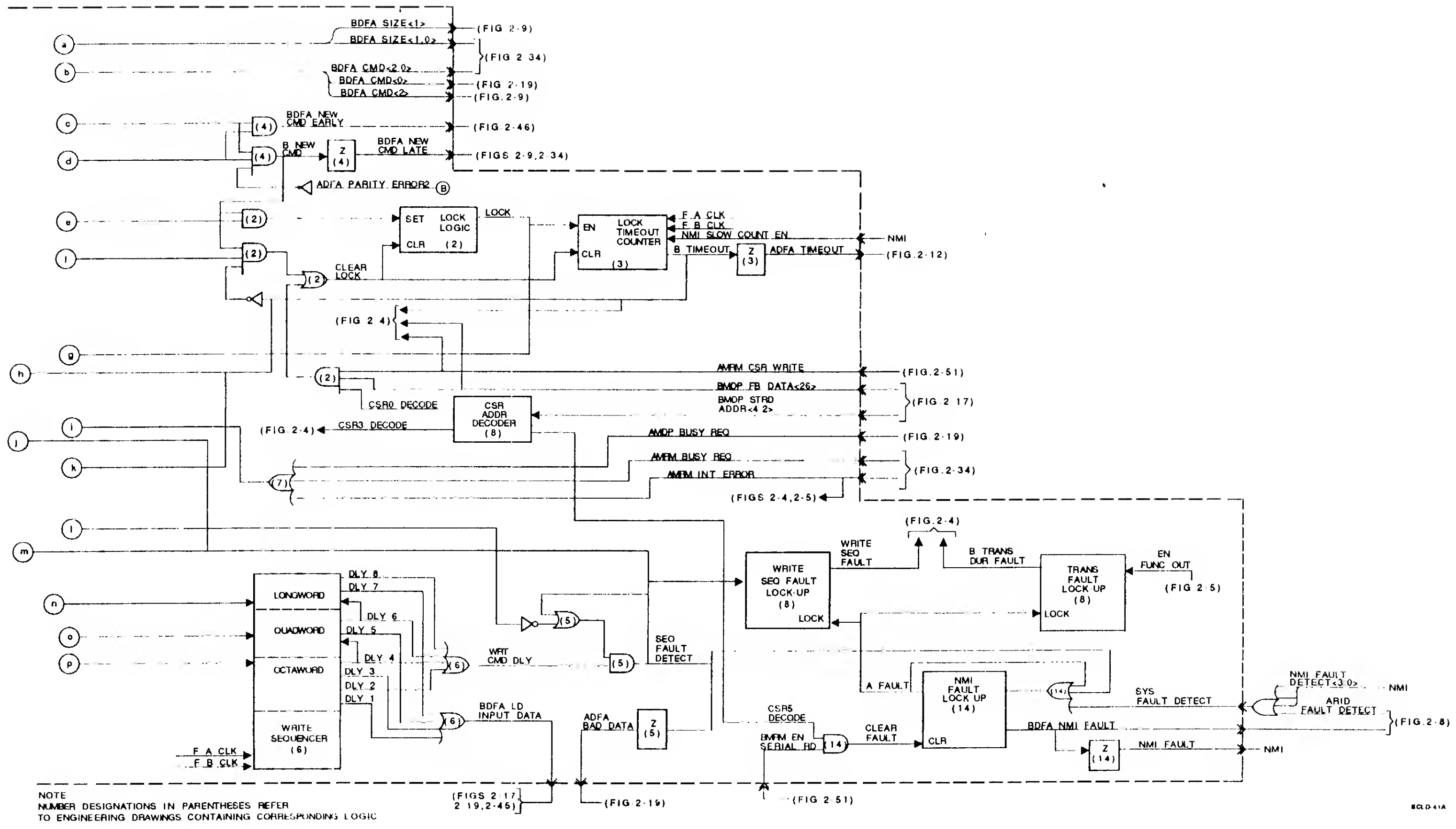


Figure 2-3 FUNK Function and Control Logic (Sheet 2 of 2)

2.3.1 Function Field Parity

The five-bit function field from the NMI (NMI FUNCTION<4:0>) is renamed and then applied to the FUNK MCA. NMI FUNCTION<3:1> is renamed XNMI FUNCTION<2:0>, and NMI FUNCTION<4>,<0> is renamed YNMI FUNCTION<1:0>.

The function bits are latched to become FUNCTION<4:0> which are then applied to a parity generator. Parity is generated on the function field and output as CTRL GEN PARITY to the ARID MCA where the function field parity is checked.

2.3.2 Function Decoder

FUNCTION<4:0> is applied to a function decoder where the commanded operation is decoded. Another input to the function decoder is address bit 29 (ADDRESS 29) which is received from the DAD MCA as ADFA DATA ADDRESS<29>. Address bit 29 specifies the commanded operation as an access to memory or to I/O space (to a CSR).

Table 2-3 lists the function bit codes and the command operations they specify.

Table 2-3 Function Codes

FUNCTION (Hex)	<4 3 2 1 0>	Command Operation
1 0	1 0 0 0 0	Read longword
1 2	1 0 0 1 0	Read octaword
1 3	1 0 0 1 1	Read hexword
1 4	1 0 1 0 0	Read longword interlocked
1 6	1 0 1 1 0	Read octaword interlocked
1 7	1 0 1 1 1	Read hexword interlocked
1 B	1 1 0 1 1	Write longword
1 F	1 1 1 1 1	Write octaword
1 8	1 1 0 0 0	Write masked longword
1 9	1 1 0 0 1	Write masked quadword
1 A	1 1 0 1 0	Write masked octaword
1 C	1 1 1 0 0	Write masked longword unlock
1 D	1 1 1 0 1	Write masked quadword unlock
1 E	1 1 1 1 0	Write masked octaword unlock
0 A	0 1 0 1 0	Read/return good data
0 E	0 1 1 1 0	Read/return bad data
0 8	0 1 0 0 0	Read/continue good data
0 C	0 1 1 0 0	Read/continue bad data
0 9	1 1 0 0 1	Write data

The decoder outputs a 2-bit size code (BDFA SIZE<1:0>) that specifies the size of the data transfer. The size code is shown in Table 2-2. BDFA SIZE<1> goes to the ARID to specify a hexword operation to the ID/mask logic. BDFA SIZE<1:0> goes to the MRM.

The decoder outputs a 3-bit command code (BDFA CMD<2:0>) that specifies the type of operation. The command code is shown in Table 2-1.

Note in Table 2-1 that bit BDFA CMD<2> specifies a read or write operation, bit BDFA CMD<1> specifies a memory or CSR access, and bit BDFA CMD<0> specifies a non-masked or masked operation. Bits BDFA CMD<2> and BDFA CMD<0> are obtained directly from the function decoder. Bit BDFA CMD<1> is derived from CSR VALID. CSR VALID is true if CSR ADDR from the DAD is asserted (the NMI address is to one of the CSRs) and the function decoder has decoded a WRITE CSR function or a READ LONGWORD function (access to CSRs must be a longword function).

BDFA CMD<0> goes to the MDBC to specify a masked operation. BDFA CMD<2> goes to the ARID to specify a read operation. BDFA CMD<2:0> goes to the MRM.

The command code (BDFA CMD<2:0>) and size code (BDFA SIZE<1:0>) specify all the functions listed in Table 2-3 except:

- A lock function
- An unlock function
- A write data cycle

These three functions are specified when the function decoder respectively asserts:

- INTERLOCK
- UNLOCK
- WRITE DATA CYCLE

NOTE

The function decoder does not decode read/return or read/continue functions as these are not commands received from the NMI. They are commands issued by the MCL (from the MRM).

The function decoder asserts other outputs, with self-explanatory mnemonics, that are used within the FUNK as discussed in the following sections.

2.3.3 NEW CMD EARLY/NEW CMD LATE

Early and late new-command signals are generated by the FUNK when a valid, error-free, new command is detected. The new-command signals are sent to the MRM as the load signals for the new command.

A new-command mux selects CSR VALID from the function decoder (true if a valid CSR access is commanded) or ADFA MEM ADDR from the decode RAM in the DFA (true if a valid memory access is commanded). The output from the new-command mux is START NEW CMD. Mux selection is made by memory/CSR address bit ADDRESS 29.

START NEW CMD is applied to two AND gates to generate BDFA NEW CMD EARLY and BDFA NEW CMD LATE. BDFA NEW CMD EARLY and BDFA NEW CMD LATE are functionally identical (BDFA NEW CMD EARLY occurs slightly before BDFA NEW CMD LATE) and are applied to different areas within the MRM. There are three conditions that will inhibit the assertion of BDFA NEW CMD EARLY and BDFA NEW CMD LATE. These are:

- FUNCTION INVALID -- asserted by the function decoder when the decoder senses a code not specifying a valid function listed in Table 2-3.
- BLOCK COMMAND -- asserted by the block-command logic when the new command is to be blocked for reasons discussed in Section 2.3.7.
- ADFA PARITY ERROR2 -- asserted by the ARID module when a parity error is detected in the ID/mask field, the function field, or the data/address field (Figure 2-7). ADFA PARITY ERROR2 inhibits only BDFA NEW CMD LATE in the FUNK, however BDFA NEW CMD EARLY is inhibited by a parity error in the MRM, thus the BDFA NEW CMD EARLY and BDFA NEW CMD LATE remain functionally identical.

2.3.4 Read Lock Function

When the function decoder senses a read-lock function, it asserts INTERLOCK. INTERLOCK is ANDed with B NEW CMD (an asserted B NEW CMD means a valid, error-free, new command) to set a lock-logic circuit. When set, the lock logic asserts LOCK which remains asserted until the lock logic is cleared by a write-unlock command or a write to CSR0.

The assertion of LOCK enables a lock-timeout counter which starts to time the interlock interval.

2.3.5 Write Unlock Function

When the function decoder senses a write-unlock function, it asserts UNLOCK. UNLOCK is ANDed with B NEW CMD to assert CLEAR LOCK (B TIMEOUT false). CLEAR LOCK clears the lock logic (thereby negating LOCK) and resets the lock-timeout counter.

Had a lock-timeout occurred, B TIMEOUT would have been true and inhibited the clearing of the lock logic by the write-unlock function. In this case the lock logic is cleared by writing bit 26 into CSR0. Writing bit 26 into CSR0 asserts CLEAR LOCK via an AND gate which is enabled by the following three inputs:

- AMRM CSR WRITE -- asserted from the MRM whenever a CSR is written.

- BMDP FB DATA<26> -- asserted from the MDP when bit 26 is true.

- CSR0 DECODE -- asserted from a CSR ADDR decoder when the three-bit address code (BMDP STRD ADDR<4:2>) received from the MDB specifies CSR0.

2.3.6 Lock-Timeout Counter

LOCK from the lock logic enables a lock-timeout counter. Once enabled, the counter is increment by a clock (NMI SLOW COUNT EN) received from the clock module via the NMI. The NMI SLOW COUNT EN clock has a period of approximately 3.3 ms.

The lock-timeout counter is reset by CLEAR LOCK. Thus, when an UNLOCK command is received and asserts CLEAR LOCK (to clear the lock logic and negate LOCK), the lock-timeout counter is reset. The negation of LOCK disables the timeout counter.

If an unlock command does not occur after three cycles of NMI SLOW COUNT EN (approximately ten to thirteen ms depending on where LOCK asserts in the NMI SLOW COUNT EN cycle), the lock-timeout counter sets and outputs B TIMEOUT. B TIMEOUT is used in the generation of BLOCK COMMAND, and to inhibit an unlock command from clearing the lock logic. It is also applied to the FUNK CSRs where it sets the CSR0 timeout bit (bit 25; Figure 2-4). B TIMEOUT is output to the ARID MCA as ADFA TIMEOUT where it causes an interrupt to the CPU if the timeout enable bit (bit 30 of CSR3) is set.

2.3.7 Block Command

BLOCK COMMAND is asserted when a condition is sensed that calls for aborting a new command. The new command is aborted by inhibiting the assertion of BDFA NEW CMD EARLY and BDFA NEW CMD LATE to the MRM.

The conditions that assert BLOCK COMMAND are described in the following:

- INTERLOCK INVALID -- indicates that a received interlock command is invalid because the memory is already locked (INTERLOCK asserts while LOCK is true). The memory will not accept a read-lock command while it is still locked from a previous read-lock command.

- UNLOCK INVALID -- indicates that a received unlock command is invalid because memory is not locked (UNLOCK asserts while LOCK is false), or because the existing lock command has timed out (UNLOCK asserts while B TIMEOUT is true). Once a read-lock command has timed-out, it cannot be unlocked by a write-unlock command.

- WRITE BUSY -- indicates that the memory is busy with a write function and cannot accept another command at this time. Similar to NMI MEMORY BUSY, however WRITE BUSY asserts sooner and therefore can block a new write command during the first data cycle of the command. NMI MEMORY BUSY does not assert soon enough to do this. Serves as a "write-busy" flag.

  WRITE BUSY is asserted when a write command is received (WRITE COMMAND asserted from function decoder) and AMRM INT ERROR, AMRM BUSY REQ, or AMDP BUSY REQ is true.

- NMI MEMORY BUSY -- indicates that the memory is busy and cannot accept another command at this time. Serves as a "read-busy" flag.

- SEQ FAULT DETECT -- indicates that a write sequence fault was detected in a prior transfer.

2.3.8 Write Sequence Fault

A write sequence fault occurs when a write command is received followed by an insufficent number of data cycles to complete the write transfer.

The FUNK MCA contains a write sequencer circuit, divided into three sections (longword, quadword, and octaword). An input to the longword section is asserted for a write longword command; an input to the quadword section is asserted for a write quadword command; and an input to the octaword section is asserted for a write octaword command.

The three inputs must first be enabled from a memory AND gate. The memory AND gate checks for a valid memory command by looking at ADFA MEM ADDR from the decode RAM (true for a memory access) and ADDRESS 29 (false for a memory access). If these two signals indicate a memory access, and the command is valid (FUNCTION INVALID, NMI MEMORY BUSY, and WRITE BUSY all false), the memory AND gate is enabled and in turn enables a path for the three write-sequencer inputs via a long AND gate, a quad AND gate, and an octa AND gate.

2.3.8.1 Write Longword to Memory -- If the function decoder detects a write longword command, it asserts WRITE LONG which enables the long gate and asserts an input (via an OR gate) to the longword section of the write sequencer. Upon sensing the input, the longword section uses the free-running A and B clocks to assert a DLY 7 output followed by a DLY 8 output. The DLY 7 and DLY 8 outputs occur during the write data cycle following the longword command.

DLY 7 asserts BDFA LD INPUT DATA to the MDBC and the MRM to load the latched write data.

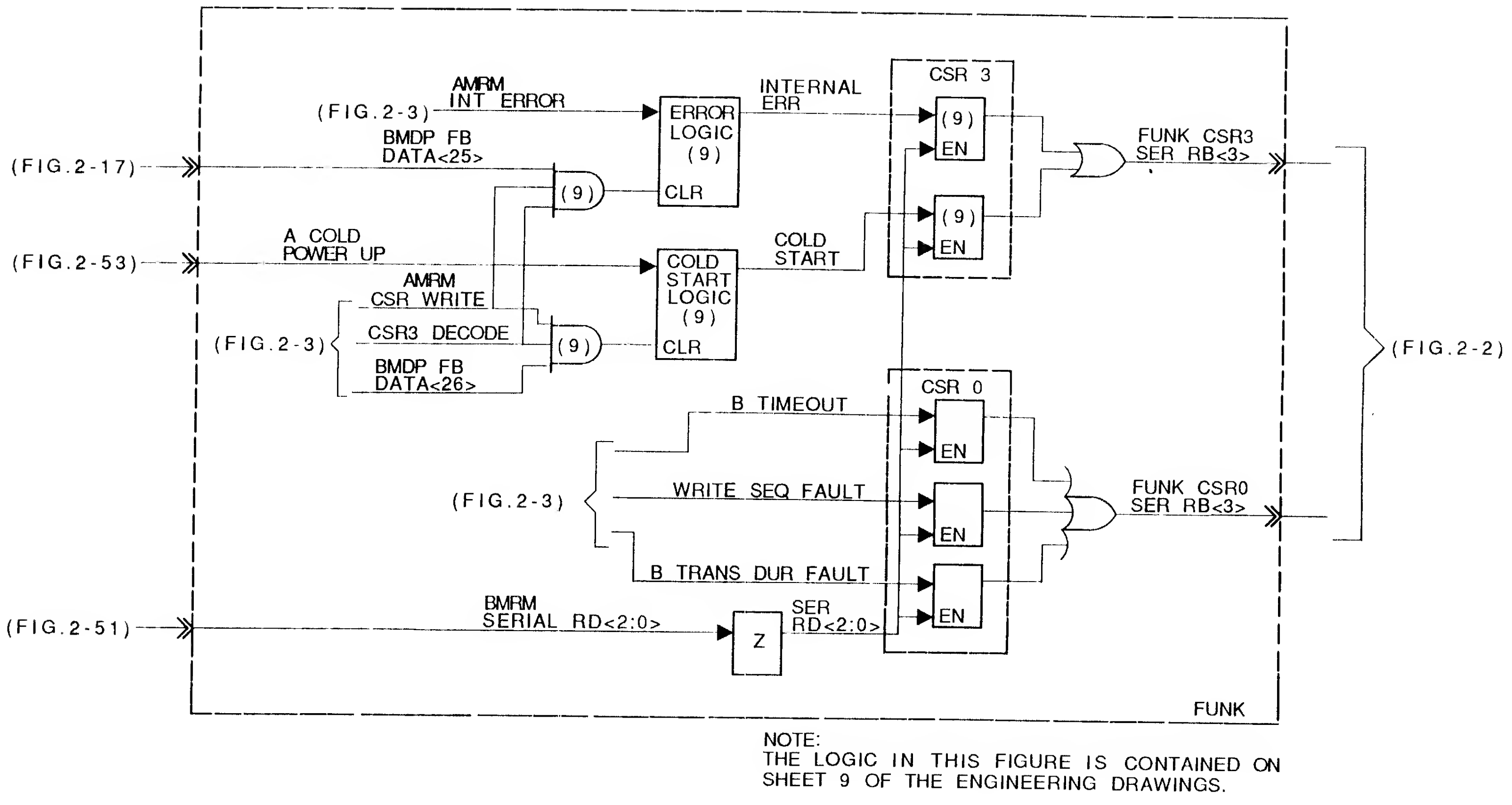


Figure 2-4 FUNK CSRs

DLY 8 asserts WRT CMD DLY to a sequence error AND gate where it is ANDed with the negated state of WRITE DATA CYCLE from the function decoder. A negated WRITE DATA CYCLE indicates that the write data cycle that should have followed the write longword command did not occur. If this is the case, SEQ FAULT DETECT asserts.

The assertion of SEQ FAULT DETECT causes the following to occur:

- Asserts BLOCK COMMAND to inhibit new transactions.
- Asserts ADFA BAD DATA to the MDBC.
- Asserts SEQ FAULT DETECT to the confirmation logic where a memory-busy code is generated to stop memory commands from NMI nexus.
- Asserts NMI FAULT from the fault logic.
- Sets the write-sequence-fault error bit in CSR0.

Note that the occurrence of a write sequence error does not cause the current write sequence to be aborted. The MCL functions to write the missing data just as though it were there. However, the bad-data bit is set and gets written into memory marking the addressed location as bad data.

2.3.8.2 Write Longword to CSR -- If the write address is to a CSR, ADFA MEM ADDR is false and ADDRESS 29 is true. This disables the long, quad, and octa AND gates, but enables a CSR AND gate. The enabling of the CSR AND gate requires that CSR ADDR from the DAD be true (the address is one of the CSRs) and that WRITE CSR from the function decoder be true. In addition, FUNCTION INVALID, NMI MEMORY BUSY, and WRITE BUSY must all be false just as in the case of a longword write to memory.

With the CSR AND gate enabled, the longword section of the write sequencer receives an input (via the OR gate) and proceeds to check for the write data cycle that should follow the command/address cycle. If the write data cycle is missing, a write-sequence error is asserted just as in the case of a longword write to memory.

2.3.8.3 Write Quadword -- Returning to the case of an access to memory, if the function decoder detects a write quadword command, it asserts WRITE QUAD which enables the quad gate and asserts an input to the quadword section of the write sequencer. Upon sensing the input, the quadword section uses the free-running A and B clocks to assert a DLY 5 output followed by a DLY 6 output. The DLY 6 output is fed back into the longword section of the write sequencer causing a DLY 7 and a DLY 8 output. All the DLY outputs occur in sequence. The DLY 5 and DLY 6 outputs occur during the first write data cycle following the quadword command. The DLY 7 and DLY 8 outputs occur during the second write data cycle following the quadword command.

DLY 5 asserts BDFA LD INPUT DATA to the MDBC and the MRM to load the latched write data of the first data cycle. DLY 6 asserts WRT CMD DLY to the sequence error AND gate to check for the presence of the first write data cycle. DLY 7 re-asserts BDFA LD INPUT DATA to the MDBC and the MRM to load the latched write data of the second write data cycle. DLY 8 re-asserts WRT CMD DLY to the sequence error AND gate to check for the presence of the second write data cycle.

If either of the write data cycles are missing, SEQ FAULT DETECT asserts and functions just as in the case of a missing write data cycle during a longword write operation.

In a multi-longword write-data transfer, a write sequence error in any of the write data cycles will hold SEQ FAULT DETECT asserted for the rest of the data cycles. This is implemented by feeding back SEQ FAULT DETECT to the sequence-error AND gate. When all the DLY signals have occurred (all the data cycles for the current transfer should have occurred), WRT CMD DLY will not assert, the sequence-error AND gate will be disabled, and the SEQ FAULT DETECT error will clear.

2.3.8.4 Write Octaword

-- If the function decoder detects a write octaword command, it asserts WRITE OCTA which enables the octa gate and asserts an input to the octaword section of the write sequencer. Upon sensing the input, the octaword section uses the free-running A and B clocks to assert DLY 1, DLY 2, DLY 3, and DLY 4 in sequence. The DLY 4 output is fed back into the quadword section causing a DLY 5 and a DLY 6 output. The DLY 6 output is fed back into the longword section causing a DLY 7 and a DLY 8 output.

The DLY 1 and DLY 2 outputs occur during the first write data cycle following the octaword command. Likewise, each write data cycle has an odd and an even numbered DLY output asserted. The odd numbered DLY output asserts BDFA LD INPUT DATA to the MDBC and the MRM to load the latched write data of the current data cycle. The even numbered DLY output asserts WRT CMD DLY to the sequence-error AND gate to check for the presence of a write data cycle command. If any of the write data cycles are missing, SEQ FAULT DETECT asserts.

Due to the feedback of SEQ FAULT DETECT, a missing data cycle early in the octaword transfer will cause SEQ FAULT DETECT and ADFA BAD DATA to stay asserted for the rest of the transfer even though the later data cycles are present.

2.3.9 NMI Faults

The FUNK receives fault-detect signals from all the NMI nexus and from the ARID, and ORs them with the FUNK write-sequence errors. The FUNK then outputs one fault line on the NMI for the entire system. The assertion of any of the fault-detect lines will assert NMI FAULT on the NMI.

NMI FAULT DETECT<3:0> is received from the other system nexus and ORed with ARID FAULT DETECT from the ARID MCA*. The result is input to the FUNK as SYS FAULT DETECT. SYS FAULT DETECT is applied (via an OR gate) to an NMI fault lock-up circuit. The NMI fault lock-up circuit asserts BDFA NMI FAULT to the ARID to latch up any fault that may have been detected by the ARID. The NMI fault lock-up circuit also asserts NMI FAULT on the NMI. A third output signal (A FAULT) is asserted and fed back to the input OR gate to lock-up the NMI fault logic and hold the three output signals asserted.

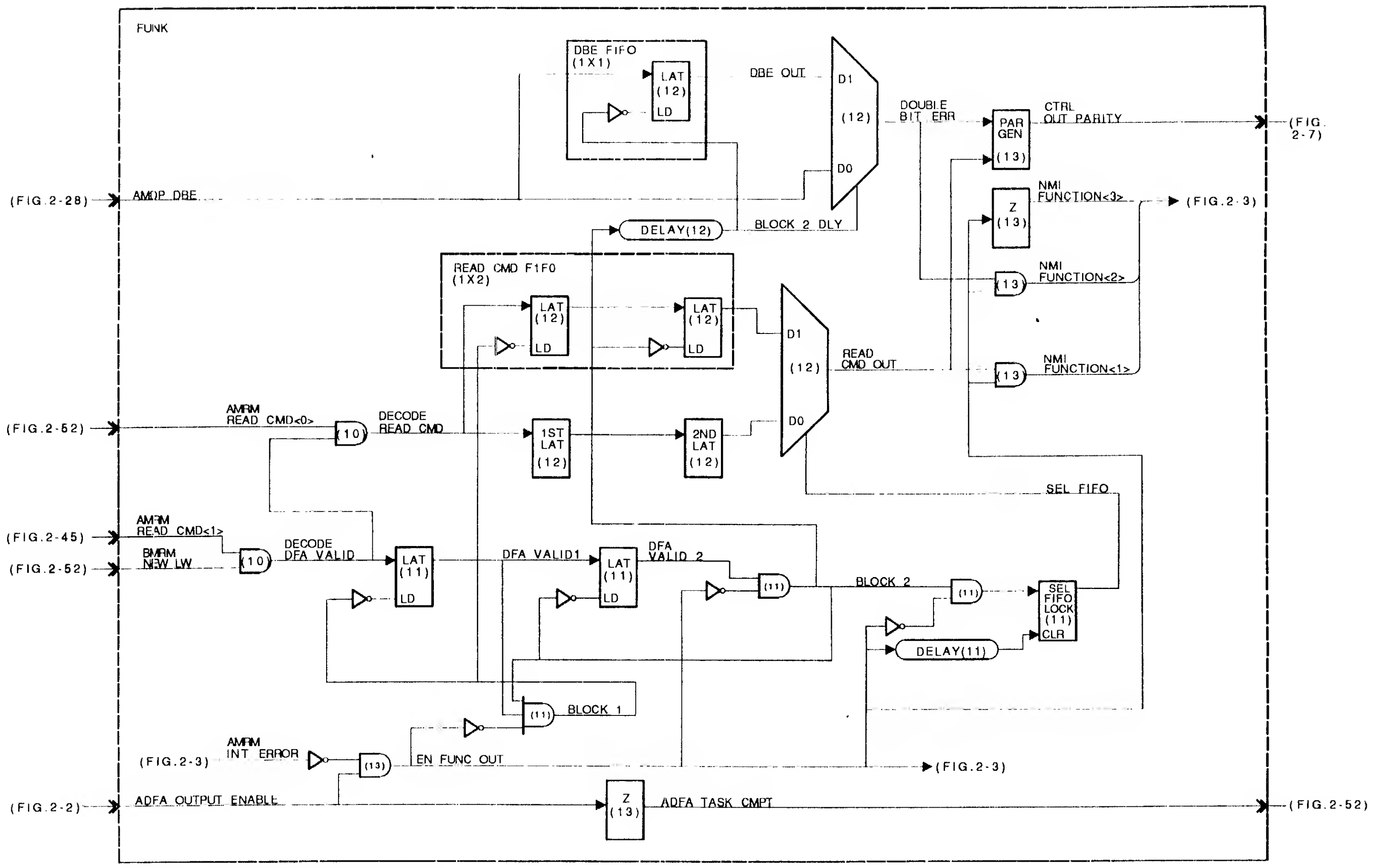
A FAULT is also applied to the lock inputs of a trans-fault lock-up circuit and a write-seq-fault lock-up circuit. The TRANS-fault lock-up circuit receives EN FUNC OUT which is derived from ADFA OUTPUT ENABLE (received from the CPU as a bus grant) when AMRM INT ERROR is false (Figure 2-5). The true state of EN FUNC OUT signifies that the MCL has the NMI and is transmitting data. Whenever EN FUNC OUT asserts, the output from the trans-fault lock-up logic (B TRANS DUR FAULT) is asserted. B TRANS DUR FAULT is applied to the FUNK CSR logic (Figure 2-4) as an error bit in CSR0. However, the assertion of EN FUNC OUT does not constitute an error unless A LOCK asserts at the same time. If EN FUNC OUT is true when A LOCK asserts, then a fault occurred while the memory was transmitting. In this case, B TRANS DUR FAULT is locked-up by A FAULT, thereby holding B TRANS DUR FAULT asserted to CSR0. When CSR0 is read, the B TRANS DUR FAULT error bit is read as being set.

The write-seq-fault lock-up circuit receives SEQ FAULT DETECT from the write sequence fault logic if a write-sequence error occurs. The write-seq-fault lock-up circuit outputs WRITE SEQ FAULT to the CSR logic as an error bit in CSR0.

SEQ FAULT DETECT is also appled to the NMI fault lock-up logic which in turn asserts its three error outputs (BDFA NMI FAULT, NMI FAULT, and A FAULT). A FAULT locks-up the NMI fault logic (holding NMI FAULT asserted on the NMI) and the write-seq-fault logic (holding WRITE SEQ FAULT asserted to CSR0). When CSR0 is read, the WRITE SEQ FAULT error bit is read as being set.

The NMI FAULT line on the NMI, and the WRITE SEQ FAULT and B TRANS DUR FAULT error bits in CSR0, are cleared by reading CSR5. The CSR

* ARID FAULT DETECT asserts when a data or a control parity error is detected.



NOTE:
NUMBER DESIGNATIONS IN PARENTHESES REFER TO ENGINEERING DRAWINGS CONTAINING CORRESPONDING LOGIC.

SCID 48

Figure 2-5 Read/Return and Read/Continue Logic

ADDR decoder decodes address input BMDP STRD ADDR<4:2> and outputs CSR5 DECODE when the CSR5 address is decoded. CSR5 DECODE is ANDed with BMRM EN SERIAL RD (true for a CSR read function) to assert CLEAR FAULT. CLEAR FAULT clears the NMI-fault lock-up logic which negates the FAULT line on the NMI and A FAULT from the NMI-fault lock-up logic. The negation of A FAULT releases the write-seq-fault lock-up logic and the trans-fault lock-up logic, thereby clearing the two fault error bits in CSR0.

2.3.10 NMI Confirmation

The MCL places two confirmation bits (NMI CONFIRMATION<1:0>) on the NMI during the second bus cycle after the command/address cycle. The confirmation bits notifies the commander that:

- The MCL does not acknowledge receipt of the command,
- The MCL accepts the command,
- The MCL is interlocked, or
- The MCL is busy.

The confirmation bit code is shown in Table 2-4.

Table 2-4 NMI Confirmation Codes

NMI CONFIRMATION <1	0>	Confirmation State
0	0	No acknowledgement
0	1	Command accepted
1	0	Interlock busy
1	1	Memory busy

The four confirmation states are described in the following.

- Memory busy -- The assertion of MEMORY BUSY asserts both NMI confirmation bits. The assertion of MEMORY BUSY requires the reception of a new command (START NEW CMD true) and that the new command be valid (FUNCTION INVALID false). In addition, one of the following three conditions must be true:

  -- NMI MEMORY BUSY from the ARID.

  -- SEQ FAULT DETECT from the write-sequence fault logic. When a write sequence error occurs, a memory-busy confirmation code is placed on the NMI to stop new commands from other NMI nexus.

  -- WRITE BUSY from the block-command logic. Used to indicate that the memory became busy during the first write-data cycle of a write transfer (NMI MEMORY BUSY not asserted yet; see Section 2.3.7).

- Interlock busy - The assertion of INTERLOCK BUSY asserts NMI CONFIRMATION<1>. The assertion of INTERLOCK BUSY requires that the access be to memory (ADFA MEM ADDR true). In addition, one of the following conditions must be true:

  -- An interlock command is received (INTERLOCK asserts) when the MCL is already locked-up (LOCK true).

  -- An unlock command tries to unlock the MCL (UNLOCK asserts) when the memory has already timed out (B TIMEOUT true).

  The negated state of COMMAND CYCLE makes NMI CONFIRMATION<0> false. COMMAND CYCLE is negated by BLOCK COMMAND during the interlock-busy state. BLOCK COMMAND is true because it is asserted by either of the two conditions that cause INTERLOCK BUSY to assert (see Section 2.3.7).

- Command accepted -- The asertion of COMMAND CYCLE asserts NMI CONFIRMATION<0>. The assertion of COMMAND CYCLE requires a new command (START NEW CMD true) and that the new command be valid (FUNCTION INVALID false), and that there be no reason to block the new command (BLOCK COMMAND false).

  The negated state of INTERLOCK BUSY makes NMI CONFIRMATION<1> false. INTERLOCK BUSY is negated because it signifies a non-valid condition.

- No acknowledgement -- The lack of any of the signals required for the assertion of COMMAND CYCLE will generate a "no ack" response (NMI CONFIRMATION<1:0> both false). In addition, a parity error detected in the ARID will also cause a no-ack response.

2.3.11 NMI DEAD

When a memory-busy state exists, the MCL is busy doing internal data processing. During this time the MCL must be isolated from the NMI so that its internal data paths may be used for the data processing. ADFA NMI DEAD performs this function. ADFA NMI DEAD is generated in the FUNK and then sent to the DAD where it is used to perform its NMI isolation task. ADFA NMI DEAD is also sent to the ARID and the MDBC, and BDFA NMI DEAD1 is sent to the MRM to inform these areas of the NMI isolation.

ADFA NMI DEAD is asserted by the assertion of NMI MEMORY BUSY so long as a write operation is not in progress (COMMAND CYCLE and WRITE DATA CYCLE false) as a write operation would require input from the NMI

2.3.12 CSRs (Figure 2-4)

Three bits of CSR0 and two bits of CSR3 reside in the FUNK. The three CSR0 bits are:

- B TIMEOUT -- from the lock timeout counter
- WRITE SEQ FAULT -- from the write-seq-fault lock-up logic
- B TRANS DUR FAULT -- from the trans-fault lock-up logic

The two CSR3 bits are:

- INTERNAL ERR -- derived from AMRM INT ERROR received from the MRM
- COLD START -- derived from A COLD POWER UP received from the MCL battery backup unit (BBU). A COLD POWER UP asserts when power is applied after a power outage which was too long for battery power to sustain memory data (battery power maintain s memory data for only ten minutes).

CSR0 and CSR3 are read out serially with the serial bit selection being made by BMRM SERIAL RD<2:0> from the MRM. The serial bits from the two CSRs are output as FUNK CSR3 SER RB<3> and FUNK CSR0 SER RB<3>.

The two CSR3 bits are cleared by commanding a CSR write (AMRM CSR WRITE asserted), selecting CSR3 (CSR3 DECODE from the CSR ADDR decoder), and asserting the appropriate feedback bit from the MDB (BMDP FB DATA<25> to clear INTERNAL ERR and BMDP FB DATA<26> to clear COLD START).

The three CSR0 bits are cleared in their respective logic areas as already described.

2.3.13 Read/Return and Read/Continue (Figure 2-5)

The MCL becomes a transmitter when it is sending read data to a commander that initiated a read transaction. When the MCL is to transmit read data, the FUNK receives a two-bit read command code from the MRM specifying the data transfer as a read/return function or a read/continue function. It also receives a double-bit error signal from the DCHK which specifies the read/return or read/continue data as being good or bad. These signals are used to generate a three-bit code for the NMI that identifies the read data transfer (see Table 2-5).

When the MCL arbitrates for the NMI bus to execute a read/return function, the NMI bus may be busy. In this case, the MCL stops the processing of the read data until the bus is won. However, before the processing is stopped, the FUNK would have received two read commands from the MRM and one double-bit error signal from the

DCHK. FIFOs are used to store the two read commands and the double-bit error signal until the MCL has won the bus.

The case of the MCL immediately getting the NMI, and the case of the MCL having to wait for the NMI, are discussed in the following sections.

2.3.13.1 MCL Immediately Gets the NMI -- Read command bits AMRM READ CMD<1:0> are received from the MRM and specify a read/return operation (first longword to DFA) or a read/continue operation (next longword to DFA). (See Table 2-6.)

Also received from the MRM is BMRM NEW LW signifying the presence of a new longword of data. BMRM NEW LW and AMRM READ CMD<1> are ANDed to produce DECODE DFA VALID. DECODE DFA VALID asserts for any data longword (read/return or read/continue).

DECODE DFA VALID and AMRM READ CMD<0> are ANDed to produce DECODE READ CMD which asserts only for read/return operations. DECODE READ CMD is pased through two signal latches and then through a read-cmd mux to become READ CMD OUT.

As the first two DECODE READ CMD bits are clocked through the two signal latches, they are also clocked into two latches in a read-cmd FIFO. The FIFO output is applied to the D1 input of the read-cmd mux.

AMDP DBE is asserted from the DCHK whenever a double-bit error is detected in a longword of read data. This could occur in a read/return longword or a read/continue longword. AMDP DBE is passed through a double-bit error mux to become DOUBLE BIT ERR.

AMDP DBE is also clocked into a DBE FIFO latch. The latch output is applied to the D1 input of the double-bit error mux.

ADFA OUTPUT ENABLE is the MCL bus grant received from the CPU. ADFA OUTPUT ENABLE asserts ADFA TASK CMPT to the MRM and EN FUNC OUT to the trans-fault lock-up logic (AMRM INT ERROR False). ADFA TASK CMPT informs the MRM that access to the NMI has been obtained and it can send more read commands and data. EN FUNC OUT enables the three-bit read code (NMI FUNCTION<3:1> onto the NMI. NMI FUNCTION<3:1> specifies what type of read function is associated with the data being transmitted. EN FUNC OUT becomes NMI FUNCTION<3>. DOUBLE BIT ERR becomes NMI FUNCTION<2> which when true, codes the function as bad data. READ CMD OUT becomes NMI FUNCTION<1> which when true, codes the function as a read-return function.

Table 2-5 lists the read-command function bit codes. Note that the three NMI function bits are included in Table 2-3 for the four read functions shown.

Table 2-5 Read Function Codes

NMI FUNCTION <3	2	1>	Command
1	0	1	Read/return good data
1	1	1	Read/return bad data
1	0	0	Read/continue good data
1	1	0	Read/continue bad data

DOUBLE BIT ERR and READ CMD OUT are also applied to a parity generator where they generate CTRL OUT PARITY. CTRL OUT PARITY is sent to the ARID where it is used to generate the function/ID parity bit (NMI FUNCT ID PARITY) which is placed on the NMI along with the read data.

2.3.13.2 MCL **Waits for the NMI** -- If the MCL has to wait for the NMI, ADFA OUTPUT ENABLE and EN FUNC OUT are false and ADFA TASK CMPT is not asserted to the MRM. The false state of ADFA TASK CMPT stops the command processing in the MRM but not before it has sent two read commands to the FUNK (and the DCHK has sent one AMDP DBE). The two read commands assert DECODE DFA VALID twice. DECODE DFA VALID loads a FIFO-control latch asserting DFA VALID1. DFA VALID1 is applied to a second FIFO-control latch asserting DFA VALID2. If EN FUNC OUT is still false at the time DFA VALID2 asserts, BLOCK 2 asserts and does the following:

- Asserts SEL FIFO to the read-cmd mux which selects the D1 input from the read-CMD FIFO.
- Latches the second stage of the read-cmd FIFO to hold the first read-command bit (read/return) at the D1 input of the read-cmd mux,
- Latches the second stage of the FIFO control to hold DFA VALID2 asserted.
- Enables the block-1 AND gate asserting BLOCK 1. BLOCK 1 latches the first stage of the FIFO to hold the second read-command bit (read/continue) and the first stage of the FIFO control to hold DFA VALID1 asserted,
- Asserts BLOCK 2 DLY to the double-bit error mux which selects the D1 input from the DBE FIFO. BLOCK 2 DLY also latches the DBE FIFO latch to hold the double-bit error bit associated with the first data longword.

When the MCL has acquired the NMI bus, ADFA OUTPUT ENABLE and then EN FUNC OUT asserts. The assertion of EN FUNC OUT negates BLOCK 2 and BLOCK 1 to release the latch stages in the read-cmd FIFO and in the FIFO control logic. The two read-command bits in the read-cmd FIFO and the double-bit-error bit in the DBE FIFO, are clocked out to the NMI through the read-cmd mux and the double-bit-err mux respectively. The negation of the mux select signals (SEL FIFO and BLOCK 2 DLY) are delayed until the two FIFOs have output their stored bits. The muxes then switch their inputs to the signal channels and normal operation is resumed.

2.3.14 MRM Hold Command (Figure 2-6)

NMI SLOW MODE is asserted by the clock module as a warning that the system is about to go into single-step operation and the normal clocks are to be stopped. When this occurs, ADFA SLOW MODE EN asserts and checks for the presence of a write command or a write data cycle. If either is present, a write function is executing in which case the FUNK asserts BDFA HOLD CMD to the MRM. BDFA HOLD CMD causes the MRM to hold but not to execute the last command.

BDFA HARBINGER is asserted when the system goes into single-step and normal clocks are stopped. BDFA HARBINGER latches ADFA SLOW MODE EN thereby keeping BDFA HOLD CMD asserted to the MRM.

When single-step mode is ended and normal clocks are resumed, BDFA HARBINGER is negated and ADFA SLOW MODE EN is unlatched. ADFA SLOW MODE EN then negates, which in turn negates BDFA HOLD CMD to the MRM and normal operation is resumed.

2.3.15 Force One Cycle (Figure 2-6)

NO NEXT CLOCK<2> is received from the NMI when the system goes into single-step operation and normal system clocks have stopped. NO NEXT CLOCK<2> asserts BLOCK A and BLOCK B to lock-up latches throughout the FUNK.

In addition, BLOCK B asserts FORCE ONE CYCLE which:

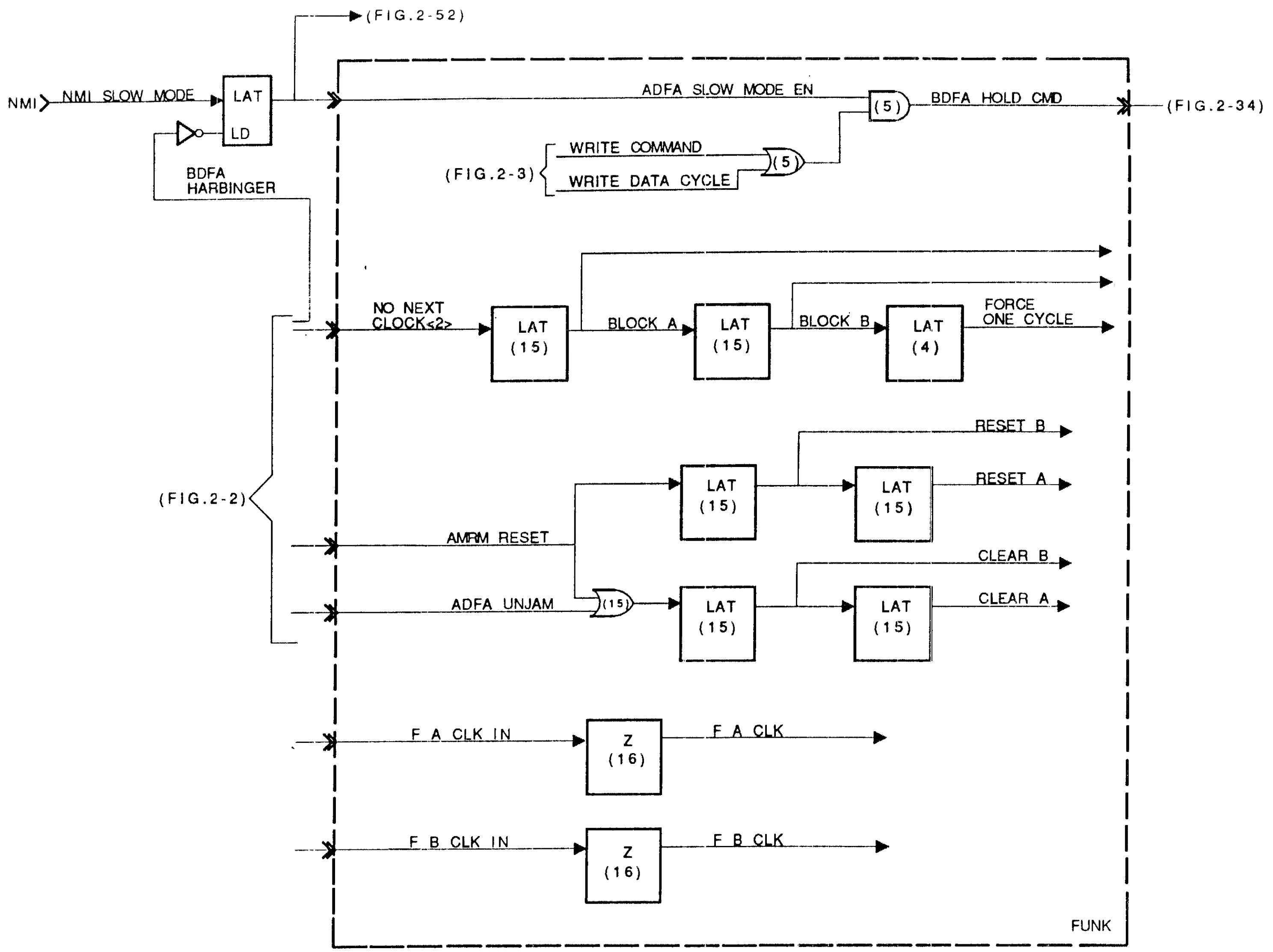
- Dlocks BDFA NEW CMD EARLY and BDFA NEW CMD LATE to the MRM.
- Locks-up the current NMI confirmation bits on the NMI.

2.4 ARBITRATION/ID (ARID) MCA

2.4.1 NMI Data Parity (Figure 2-7)

NMI DATA PARITY is a bi-directional signal line that carries the data parity bits (and the address parity bits) associated with the data being transferred between the NMI and memory.

2.4.1.1 **Parity In** -- When an input, NMI DATA PARITY is the address parity bit for a command/address cycle (read or write), or the data parity bit for a write data cycle.



NOTE:
NUMBER DESIGNATIONS IN PARENTHESES
REFER TO ENGINEERING DRAWINGS
CONTAINING CORRESPONDING LOGIC.

SCLD-46

Figure 2-6 Clock and Command Control Logic

NMI DATA PARITY is latched to become A DATA PARITY, and then output from the ARID into a parity error detector. Also applied to the parity error detector is ADFA GEN PARITY<3:0> from the DAD. ADFA GEN PARITY<3:0> are the byte parity bits generated on the address/data as it is received into the DAD from the NMI. The parity error detector compares ADFA GEN PARITY<3:0> with A DATA PARITY and if a parity error is detected, asserts DATA PARITY ERR. DATA PARITY ERR asserts ADFA PARITY ERROR1 to the MDBC, and ADFA PARITY ERROR2 to the FUNK and MRM. In the MDBC, ADFA PARITY ERROR1 asserts the bad-data bit for the associated longword during write-data cycles. In the FUNK, ADFA PARITY ERROR2 aborts the operation and forces a no-response confirmation code on the NMI during the command/address cycle.

When ADFA NMI DEAD is false, DATA PARITY ERR is applied to the ARID fault-detect logic and (via the fault-detect logic) to CSR0 in the CSR logic.

When ADFA NMI DEAD is true, DATA PARITY ERR does not cause an ARID fault. In the DAD, ADFA NMI DEAD is asserted when the wrap-data path is being used and NMI inputs are blocked. Thus, the parity bit from the NMI (NMI DATA PARITY) would not be related to wrapped data parity and the parity check is invalid.

2.4.1.2 **Parity Out** -- When an output, NMI DATA PARITY is the data parity bit for each longword of read data transmitted from memory to the NMI.

BDFA OUT PARITY<3:0> are byte parity bits generated on the read data in the DAD. BDFA OUT PARITY<3:0> is applied to a parity generator which outputs a composite parity bit (PAR30) which is then applied to an XOR gate. AMDP DATA SBE is also applied to the XOR gate to correct the parity whenever a single-bit error occurs. Parity correction is required because parity is generated in the DAD before single-bit error correction occurs. Hence, whenever a single-bit error is corrected, the parity generated is not right for the corrected data. AMDP DATA SBE corrects this by flipping PAR30.

The output from the XOR gate (OUTPAR) is applied to a latch. When HOLD DATA PARITY is false, OUTPAR is loaded into the latch and then gated onto the NMI DATA PARITY line by the true state of OUTPUT.

OUTPUT is true and HOLD DATA PARITY is false when memory has use of the NMI. If memory is arbitrating for but has not yet won the NMI, OUTPUT is false to isolate the memory from the NMI, and HOLD DATA PARITY is true to latch the OUTPAR parity bit until the NMI bus is won.

2.4.2 NMI Function/ID Parity (Figure 2-7)

NMI FUNCT ID PARITY is a bi-directional signal line that carries the parity bit for the function and commander ID (or write mask).

2.4.2.1 Parity In -- When an input, NMI FUNCT ID PARITY is the function and ID parity bit for a command/address cycle (read or write), and the function and mask parity bit for a write data cycle.

NMI FUNCT ID PARITY is applied to a parity error detector where the function parity and the ID (or mask) parity are checked. The ID (or mask) is input from the NMI as NMI ID MASK<3:0> and then latched to become ID<3:0>. ID<3:0> is applied to a parity generator which generates the parity bit, ID XOR. ID XOR is applied to the parity error detector along with function parity bit CTRL GEN PARITY generated in the FUNK. CTRL GEN PARITY and ID XOR are compared with NMI FUNCT ID PARITY and if a parity error is detected, the parity error detector outputs CTRL PAR ERR.

CTRL PAR ERR becomes CTRL PARITY ERR which is then output from the ARID. Outside the ARID, CTRL PARITY ERR is ORed with DATA PARITY ERR, hence, it too asserts ADFA PARITY ERROR1 and ADFA PARITY ERROR2.

CTRL PAR ERR is also applied to the fault-detect logic and (via the fault-detect logic) to CSR0 in the CSR logic.

2.4.2.2 Parity Out -- When an output, NMI FUNCT ID PARITY is the parity bit for the function and commander ID during a read/return or a read/continue operation.

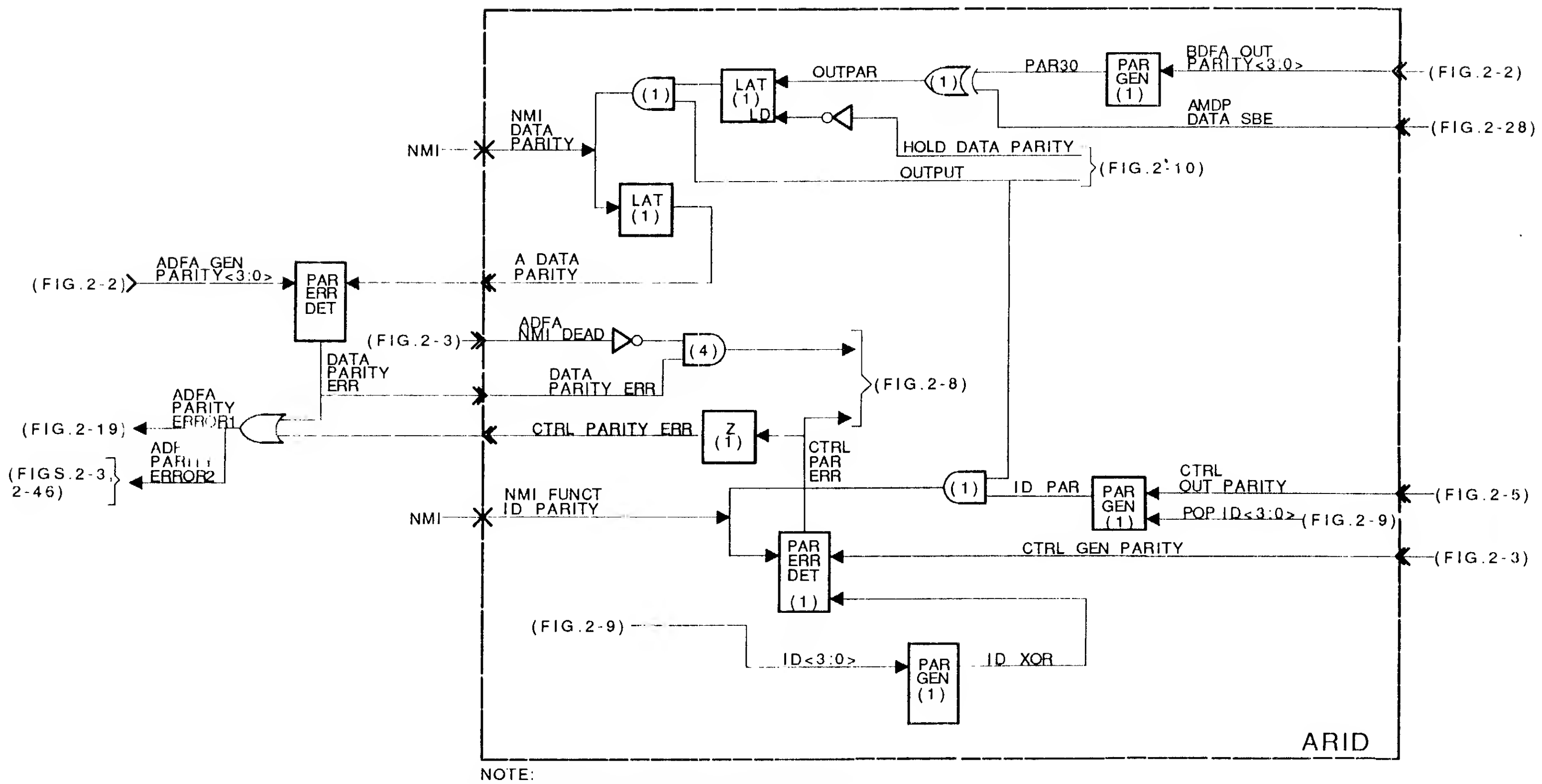
CTRL OUT PARITY is the read command parity bit generated in the FUNK. CTRL OUT PARITY is applied to a parity generator along with the commander ID (POP ID<3:0>) from the ID/mask logic. The parity generator output (ID PAR) is a composite parity bit for the command and ID.

ID PAR is gated onto the NMI FUNCT ID PARITY line by the true state of OUTPUT. OUTPUT is true when memory has use of the NMI.

2.4.3 Fault Detect (Figure 2-8)

ARID FAULT DETECT is asserted to the FUNK MCA whenever the ARID detects a parity error on the incoming data.

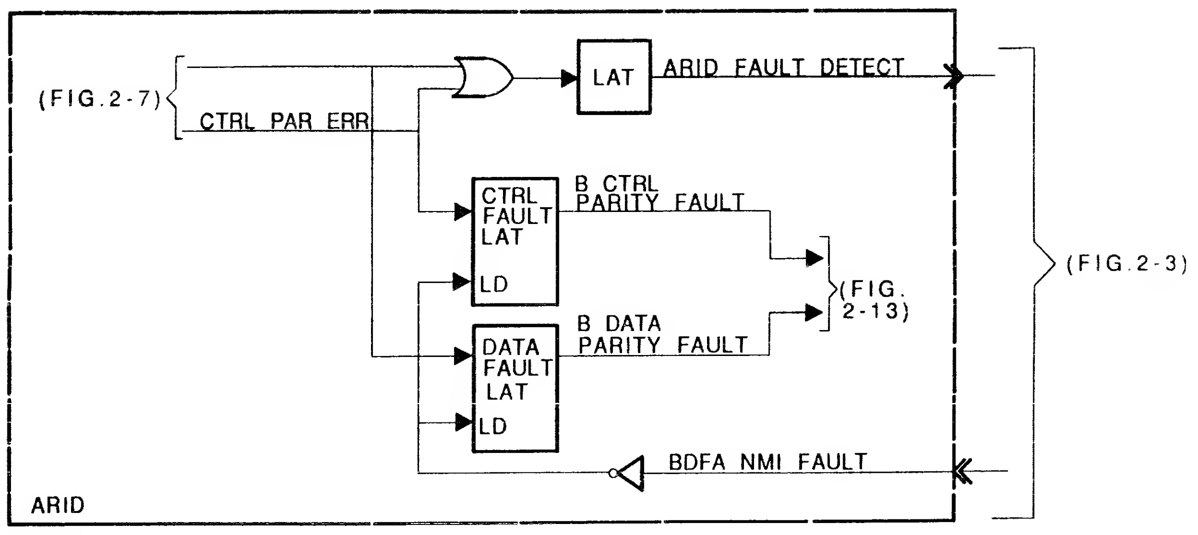
A data parity error signal is received from the parity generation and checking logic when a parity error is detected on the address or write data being received from the NMI. CTRL PAR ERR is received from the parity generation and checking logic when a parity error is detected on the function and ID (or write mask) being received from the NMI. Either parity error will assert ARID FAULT DETECT to the FUNK MCA.



NOTE:
NUMBER DESIGNATIONS IN PARENTHESES
REFER TO ENGINEERING DRAWINGS
CONTAINING CORRESPONDING LOGIC.

SCLD-52

Figure 2-7 Parity Generation and Checking



NOTE:
THE LOGIC IN THIS FIGURE IS CONTAINED ON SHEET 4 OF THE ENGINEERING DRAWINGS.

SCLD 51

Figure 2-8 Fault Detect Logic

CTRL PAR ERR is applied to a CTRL-fault latch which outputs B CTRL PARITY FAULT. Likewise, a data parity error asserts the input to a data-fault latch which outputs B DATA PARITY FAULT. BDFA NMI FAULT from the FUNK, latches the two fault latches thereby holding any error they may contain until the fault is cleared.

The latched fault-errors (B CTRL PARITY FAULT and B DATA PARITY FAULT) are applied to CSR0 in the ARID CSR logic.

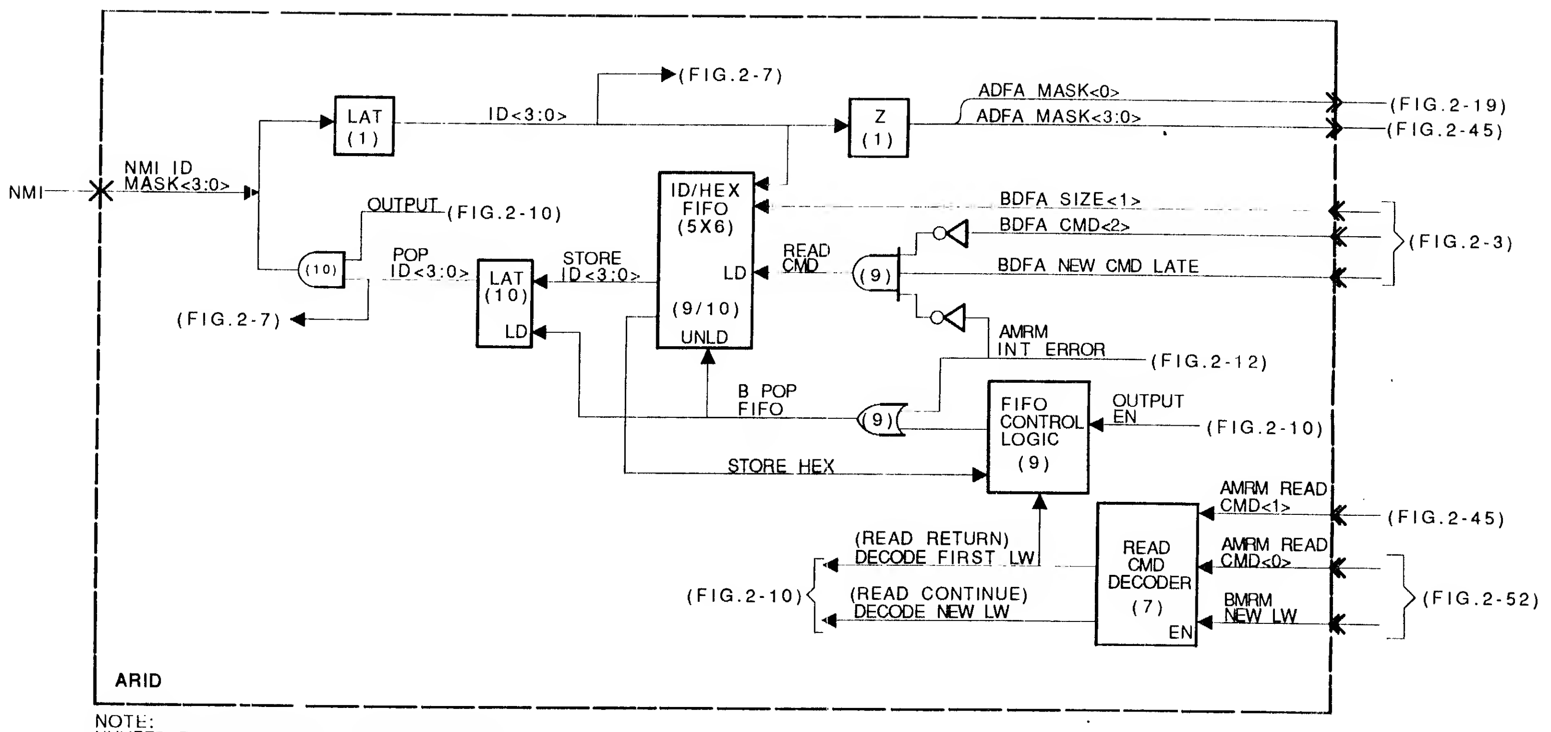
2.4.4 NMI ID/Mask (Figure 2-9)

2.4.4.1 ID/Mask In -- When an input, NMI ID MASK<3:0> is the ID of the commander nexus for a command/address cycle (read or write), or the write mask for a write data cycle. NMI ID MASK<3:0> is latched to become ID<3:0> which is applied to the parity generation and checking logic for a parity check.

A. Commander ID

In a command/address cycle, ID<3:0> (commander ID) is applied to the input of a six-deep ID/hex FIFO. The commander ID is loaded into the FIFO by READ CMD which asserts for every new, error-free, read command received by the memory (BDFA CMD<2> is false for read commands; see Table 2-1).

Also entered into the FIFO is a hex flag (BDFA SIZE<1>) received from the FUNK. BDFA SIZE<1> is asserted for hex reads (see Table 2-2).



NOTE:
NUMBER DESIGNATIONS IN PARENTHESES
REFER TO ENGINEERING DRAWINGS CONTAINING CORRESPONDING LOGIC.

SCLD-50

Figure 2-9 ID/Mask Logic

The commander ID and its associated hex flag is stored in the FIFO until the memory transmits the requested read data. At this time, the FIFO is unloaded to supply the ID to go with the read data.

The FIFO can store up to six read commands, after which the memory enters the "busy" state and asserts BUSY on the NMI.

B. Write Mask

In a write data cycle, ID<3:0> (write mask) is output to the MRM as ADFA MASK<3:0> where it functions as the write mask. In addition, ADFA MASK<0> is sent to the MDBC to specify a mask operation.

2.4.4.2 ID Out -- When an output, NMI ID MASK<3:0> is the ID of the commander that requested the read data.

The ID/hex FIFO is unloaded for the read/return commands of a longword read, an octaword read, and the first half of a hexword read. At these times, the FIFO is unloaded to supply the commander ID that goes with the read data. The hex flag associated with the read/return function is unloaded along with the ID. The FIFO is not unloaded for read/continue commands because the same ID is used.

A special case occurs during the second octaword read of a hexword read when a read/return command will occur but the FIFO is not to be unloaded (same ID for both octawords). The hex flag signals this condition and inhibits the unloading of the FIFO. The unloading of the ID/hex FIFO is controlled by the FIFO control logic as described in the following.

BMRM NEW LW is asserted from the MRM for every new longword of read data. BMRM NEW LW enables a read-command decoder which decodes the two-bit read commands (AMRM READ<1:0>) generated by the MRM. The read decoder asserts DECODE FIRST LW for a read/return operation and DECODE NEW LW for a read/continue operation.

DECODE FIRST LW and DECODE NEW LW are sent to the arbitration logic where they are used to acquire (and hold) the NMI for the read data.

In addition, DECODE FIRST LW is applied to the FIFO control logic causing it to assert B POP FIFO (via an OR gate). B POP FIFO unloads the stored ID (STORE ID<3:0>) and loads it into a latch where it becomes POP ID<3:0>. If the memory has won the bus (OUTPUT true), the commander ID is gated to the NMI as NMI ID MASK<3:0>.

When B POP FIFO negates, POP ID<3:0> is latched to provide the ID for any read/continue operations that may follow.

POP ID<3:0> is also applied to the parity generation and checking logic for the generation of the function/ID parity.

As the ID was unloaded from the FIFO, its associated hex flag (STORE HEX) was also unloaded and applied to the FIFO control logic. If the FIFO control logic senses a hexword read operation (STORE HEX true), it will not assert B POP FIFO on the next assertion of DECODE FIRST LW because the next four longwords will be going to the same commander hence the same ID is used.

If the memory does not immediately win the NMI bus, two read commands may have been issued by the MRM before it halts processing the read data. If these two commands are both longword reads, the read command decoder will have issued a second DECODE FIRST LW signal to the FIFO control logic. The FIFO control logic monitors OUTPUT EN to determine if the memory has won the NMI bus. If it senses OUTPUT EN to be false (memory has not won the NMI) it holds the second DECODE FIRST LW signal until OUTPUT EN asserts. When OUTPUT EN asserts, the control logic asserts B POP FIFO to unload the new ID associated with the second longword-read operation.

Note that a memory internal error (AMRM INT ERROR) inhibits the FIFO load command (READ CMD) and asserts B POP FIFO. This results in a flush of the FIFO by stopping any new IDs from being loaded, and unloading all the stored IDs.

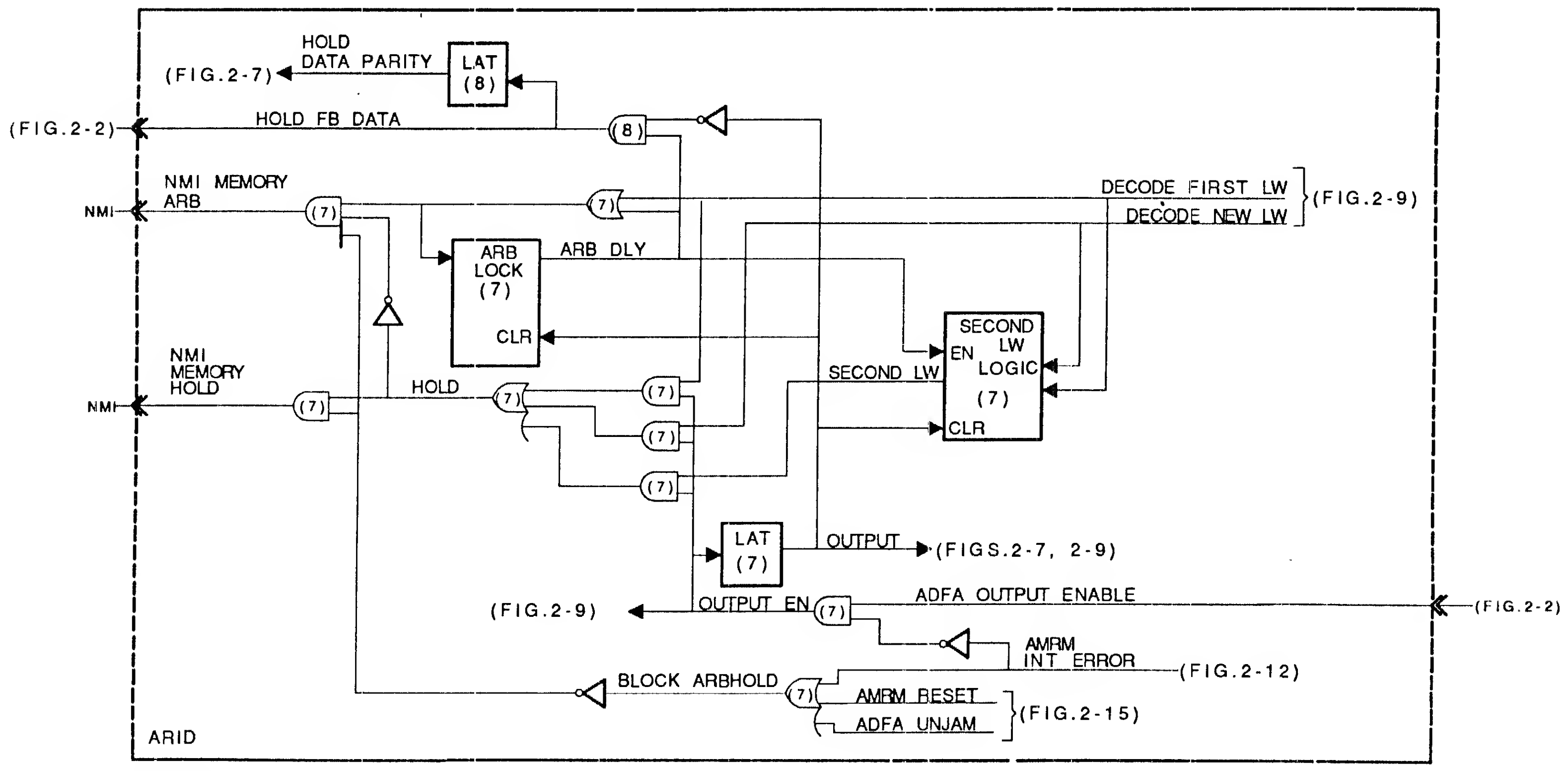
2.4.5 Arbitration/Hold Logic

The arbitration/hold logic places an arbitration request on the NMI bus when the memory has read data to be transmitted. When the memory has won the bus, the arbitration request is removed and the read data is transmitted. If the read transmission is an octaword read or a hexword read, the logic places a hold signal on the NMI after the arbitration request is removed, to hold the bus while the memory transmits the rest of the read data.

Several cases are considered which use the block diagram of Figure 2-10. The cases are divided into two general categories:

- Memory arbitrates and gets the bus right away
- Memory arbitrates but does not get the bus right away

2.4.5.1 Memory Gets the Bus Right Away - Longword Read -- DECODE FIRST LW asserts from the ID/mask logic signifying a read/return operation. DECODE FIRST LW asserts NMI MEMORY ARB on the NMI via an OR gate and an enabled AND gate. Memory is granted the bus causing ADFA OUTPUT ENABLE to assert which in turn asserts OUTPUT EN (providing there is no internal error) and OUTPUT. ADFA OUTPUT ENABLE is also applied to the DAD where it functions to transfer the read data to the NMI during the next bus cycle. It is not necessary to hold the NMI.



NOTE:
NUMBER DESIGNATIONS IN PARENTHESES
REFER TO ENGINEERING DRAWINGS
CONTAINING CORRESPONDING LOGIC.

SCLD 39

Figure 2-10 Arbitration/Hold Logic

Note that if an internal error is detected, BLOCK ARBHOLD is asserted which prevents the memory from arbitrating for the NMI. With AMRM INT ERROR true, the read data is unreliable and is not placed on the NMI. BLOCK ARBHOLD is also asserted by AMRM RESET or ADFA UNJAM.

2.4.5.2 Memory Gets the Bus Right Away - Octaword Read (Figure 2-11) -- DECODE FIRST LW asserts NMI MEMORY ARB on the NMI. Memory is granted the bus causing ADFA OUTPUT ENABLE, OUTPUT EN, and OUTPUT to assert. On the next cycle (the first data cycle), while the first longword of read data is transferring to the NMI, DECODE NEW LW asserts for the first read/continue operation. The true states of DECODE NEW LW and OUTPUT EN enable an AND gate which asserts HOLD. HOLD asserts NMI MEMORY HOLD (providing BLOCK ARBHOLD is false) to hold the NMI for the next cycle (the second data cycle) during which the second longword is transferred to the NMI. (The true state of NMI MEMORY HOLD causes the NMI arbitrator in the CPU to maintain the bus grant to the memory thereby keeping OUTPUT EN and OUTPUT asserted.)

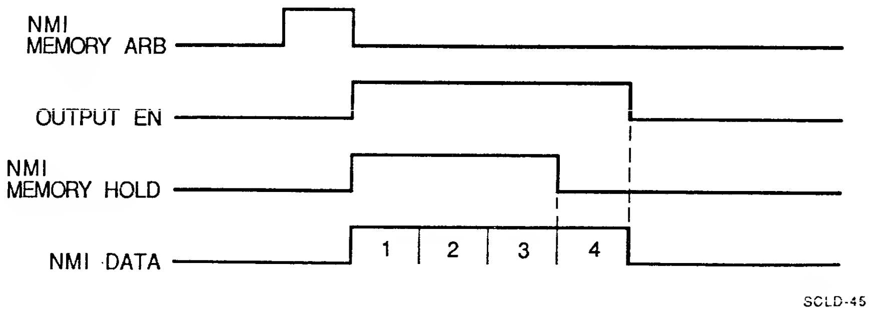


Figure 2-11 NMI Arbitration/Hold Timing

The next two assertions of DECODE NEW LW continue to keep NMI MEMORY HOLD asserted so that the last two data cycles can complete. Note in Figure 2-11 that NMI MEMORY HOLD is asserted during the first three data cycles but not during the fourth. NMI MEMORY HOLD is asserted only when the NMI is needed for the following cycle.

2.4.5.3 Memory Gets the Bus Right Away - Longword Read Back-to-Back With Another Read Function -- DECODE FIRST LW asserts NMI MEMORY ARB on the NMI. OUTPUT EN and OUTPUT assert as the bus is won. On the next cycle, as the read data from the first longword read is being transferred to the NMI, a second read function occurs and asserts DECODE FIRST LW again. The true states of DECODE FIRST LW and OUTPUT EN enable an AND gate which asserts HOLD. HOLD inhibits the re-assertion of NMI MEMORY ARB (due to the second assertion of DECODE FIRST LW) and asserts NMI MEMORY HOLD to hold the NMI for the next cycle during which the read data from the second read function is transferred to the NMI.

2.4.5.4 Memory Gets the Bus Right Away - Hexword Read -- A hexword read is two octaword reads as described in Section 2.4.5.2. MCL requirements dictate that at least one bus cycle elapse between the two octawords of a hexword read. Hence, re-arbitration for the NMI must occur for the second octaword read.

2.4.5.5 Memory Does Not Get the Bus Right Away - Longword Read -- DECODE FIRST LW asserts NMI MEMORY ARB on the NMI. An ARB lock circuit senses the arbitration request and asserts ARB DLY. The ARB lock circuit is usually held reset by OUTPUT, however, as the memory does not have the bus, OUTPUT is false. On the next cycle, DECODE FIRST LW negates but ARB DLY keeps NMI MEMORY ARB asserted on the NMI as long as necessary.

By means of an enabled AND gate, ARB DLY asserts HOLD FB DATA and HOLD DATA PARITY. HOLD FB DATA ia sent to the DAD where it latches up the read data until the bus is won. HOLD DATA PARITY is sent to the parity generation and checking section of the ARID where it does the same thing for the data parity.

When the memory wins the NMI bus, OUTPUT is asserted and clears the ARB lock circuit, thereby removing the arbitration request from the NMI. In addition, the assertion of OUTPUT negates HOLD FB DATA and HOLD DATA PARITY, thereby allowing the read data and its associated parity to transfer out to the NMI on the next cycle.

2.4.5.6 Memory Does not Get the Bus Right Away - Two Longword Reads Back-to-Back or an Octaword Read -- If the memory arbitrates for, but does not get the bus right away, the MRM could have issued two read commands before it halts its processing of the read operation. This could be two back-to-back longword reads or a read/return and the first read/continue of an octaword read. The following discusses these two situations.

DECODE FIRST LW asserts NMI MEMORY ARB on the NMI but no memory grant is received from the NMI arbitrator (in the CPU) so OUTPUT EN and OUTPUT are false. NMI MEMORY ARB is locked up by ARB DLY which is also asserting HOLD FB DATA and HOLD DATA PARITY to latch the read data and the read data parity in the DAD.

The assertion of ARB DLY enables a second-longword logic block which monitors the DECODE FIRST LW and DECODE NEW LW signals. If the operation that initiated the arbitration was an octaword read, the second-longword logic will sense the assertion of DECODE NEW LW caused by the first read/continue of the octaword read. The second-longword logic responds to DECODE NEW LW by asserting SECOND LW to an AND gate. When the bus is won and OUTPUT EN asserts, the AND gate is enabled. This results in the assertion of HOLD and NMI MEMORY HOLD to hold the bus for the transfer of the read/continue data on the next cycle. Winning the bus also asserts OUTPUT which clears the second-longword logic thereby negating SECOND LW. The assertion of DECODE NEW LW (due to the third and fourth read longwords) functions to keep NMI MEMORY HOLD asserted.

Had this been two back-to-back longword reads, the first longword read would have initiated the bus arbitration, asserted ARB DLY, and enabled the second-longword logic. The second longword read would have re-asserted DECODE FIRST LW which would have been sensed by the enabled second-longword logic. The logic would then output SECOND LW to await memory winning the bus as already discussed.

2.4.6 Interrupts (Figure 2-12)

Any of the following four errors may cause the ARID to issue an interrupt to the CPU.

- An internal error
- A lock timeout
- A single-bit error
- A double-bit error

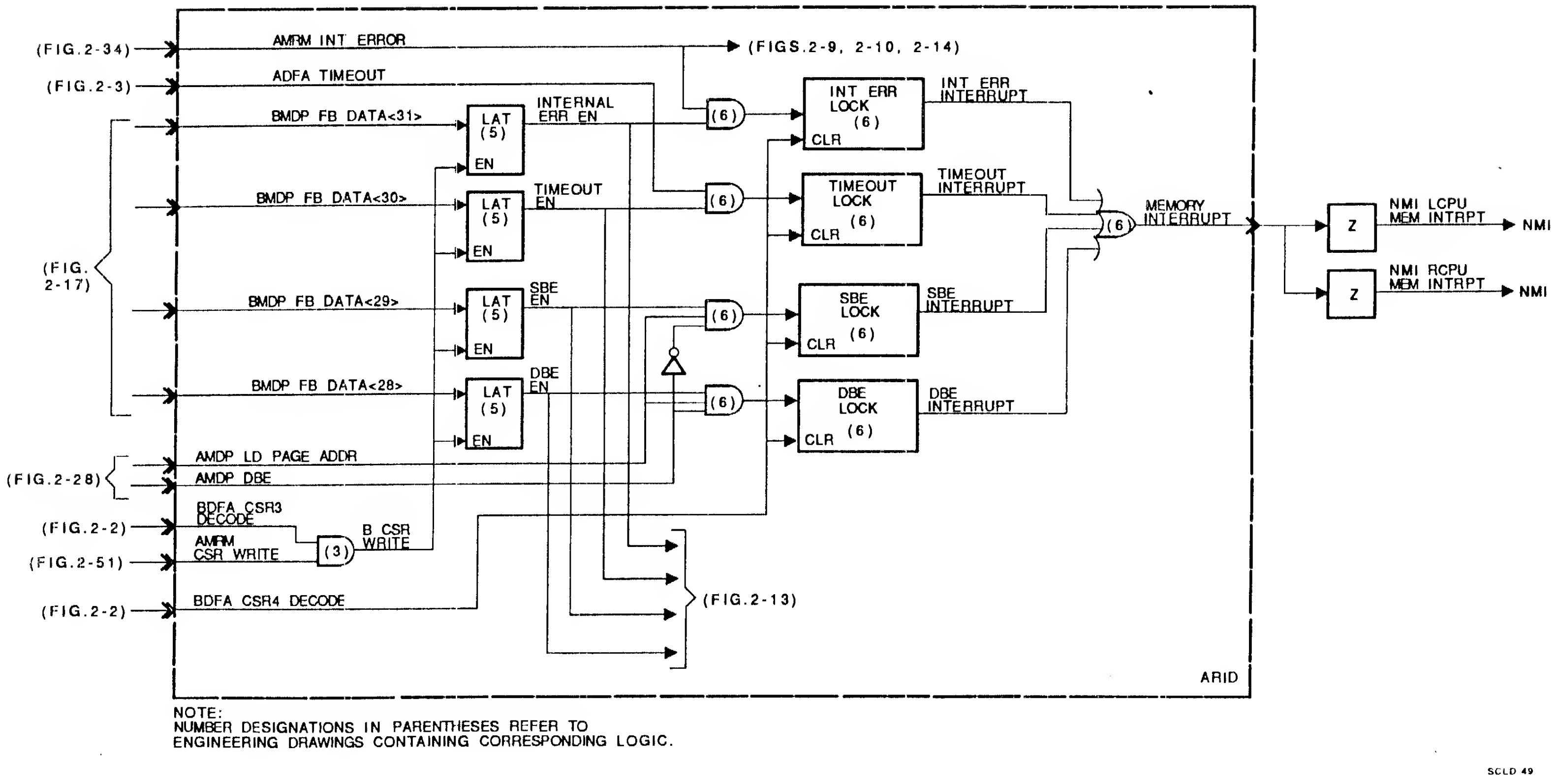
Before any of these errors can cause an interrupt, they must be interrupt-enabled by writing CSR3 with the appropriate bits.

To write CSR3, BDFA CSR3 DECODE is asserted from the DAD indicating that CSR3 is being addressed. AMRM CSR WRITE is asserted from the MRM indicating a CSR write command. With these two inputs true, B CSR WRITE asserts and load-enables four latches. Each latch receives one of four feedback bits (BMDP FB DATA<31:28>) from the MDP. BMDP FB DATA<31:28> respectively assert INTERNAL ERR EN, TIMEOUT EN, SBE EN, and DBE EN when a 1 is written into the respective latch. After the latches are written, the load enable signal (B CSR WRITE) is negated to lock-up the enable signals that were asserted.

The four interrupt enable signals are applied to CSR3 in the CSR logic.

Interrupt enable signal INTERNAL ERR EN enables an AND gate that allows the AMRM INT ERROR input from the MRM to assert an input to an INT ERR lock. The INT ERR lock outputs INT ERR INTERRUPT which in turn asserts MEMORY INTERRUPT as an ARID output. MEMORY INTERRUPT then aserts NMI LCPU MEM INTRPT to interrupt the left CPU, and NMI RCPU MEM INTRPT to interrupt the right CPU.

Likewise, TIMEOUT EN enables an AND gate allowing ADFA TIMEOUT from the FUNK to assert TIMEOUT INTERRUPT from the timeout lock. TIMEOUT INTERRUPT asserts MEMORY INTERRUPT which in turn asserts interrupt signals to the left and right CPUs.



NOTE:
NUMBER DESIGNATIONS IN PARENTHESES REFER TO ENGINEERING DRAWINGS CONTAINING CORRESPONDING LOGIC.

Figure 2-12 Interrupt Logic

SBE EN and DBE EN enable similar channels which allow respective assertions of SBE INTERRUPT or DBE INTERRUPT. SBE INTERRUPT or DBE INTERRUPT asserts MEMORY INTERRUPT which generates memory interrupts to the CPUs.

Note that AMDP LD PAGE ADDR must be asserted from the DCHK in order for a single-bit error or a double-bit error to cause an interrupt. AMDP LD PAGE ADDR is asserted by the DCHK whenever there is a bit error in the read data. When AMDP LD PAGE ADDR is asserted, AMDP DBE from the DCHK is checked to determine if the error is a single-bit or a double-bit error. The assertion of a double-bit error from the DCHK (AMDP DBE true) will assert an interrupt signal into the double-bit error interrupt channel, while the lack of a double-bit error (AMDP DBE false) will assert an interrupt signal into the single-bit error interrupt channel.

A locked-up interrupt signal is cleared by reading CSR4. When the DAD senses that CSR4 is being read, it asserts BDAF CSR4 DECODE to the ARID. BDFA CSR4 DECODE is applied to the four lock circuits where it clears the interrupt signal.

2.4.7 CSRs (Figure 2-13)

The ARID contains two bits of CSR0 and four bits of CSR3.

2.4.7.1 CSR0 -- The two CSR0 bits are the function/ID/mask parity fault bit (B CTRL PARITY FAULT) and the data/address parity fault bit (B DATA PARITY FAULT) received from the fault-detect logic.

Fault bits B CTRL PARITY FAULT and B DATA PARITY FAULT are read out serially when the three-bit read code from the MRM (BMRM SERIAL RD<2:0>) enables the fault bits out of the ARID as ARID CSR0 SER RB<3>. ARID CSR0 SER RB<3> is sent to the CSR logic in the DAD.

2.4.7.2 CSR3 -- The CSR3 bits are the four interrupt-enable bits received from the interrupt logic. BMRM SERIAL RD<2:0> from the MRM enables the four bits to be read out serially as ARID CSR3 SER RB<3>. ARID CSR3 SER RB<3> is sent to the CSR logic in the DAD.

2.4.8 Memory Busy (Figure 2-14)

The ARID asserts memory-busy on the NMI when the memory is busy and cannot accept a new command. The ARID senses a memory-busy condition by any of the following three inputs:

- AMRM BUSY REQ from the MRM
- AMRM INT ERROR from the MRM *
- AMDP BUSY REQ from the MDBC *

Any of these three inputs will assert NMI MEM BUSY ARB on the NMI and NMI MEMORY BUSY on the NMI and to the FUNK MCA.

NMI MEMORY BUSY notifies all NMI nexus not to send any new commands to the memory. NMI MEM BUSY ARB is an identical but separate NMI line that connects only to the arbitration logic in the CPU. NMI MEM BUSY ARB disables the arbitration logic in the CPU so that no nexus can arbitrate for the memory. A dedicated line is used for NMI MEM BUSY ARB to reduce signal loading so that the CPU arbitration logic will be notified of the memory-busy condition as quickly as possible.

An NMI requirement is that the memory asserts busy on the NMI, for a minimum of 2 bus cycles+. Delay logic is used in the memory-busy logic to meet this requirement. If the MRM or the MDBC asserts a busy request, it asserts the output of an OR gate which in turn asserts NMI MEMORY BUSY and NMI MEM BUSY ARB. On the next cycle (50 ns later) an AND gate receives a delayed output from the OR gate. If the memory busy request is asserted for only one cycle, the AND gate is enabled and holds busy asserted on the NMI for one more cycle.

When the busy request is asserted for more than two cycles, a second 50 ns delay disables the AND gate thereby making the NMI busy signal a direct function of the busy request.

2.4.9 Clocks and Clock Control (Figure 2-15)

NO NEXT CLOCK<3> is asserted from the DAD during single-step operation. It, in turn, asserts BLOCK A and BLOCK B to stop various clocks throughout the ARID.

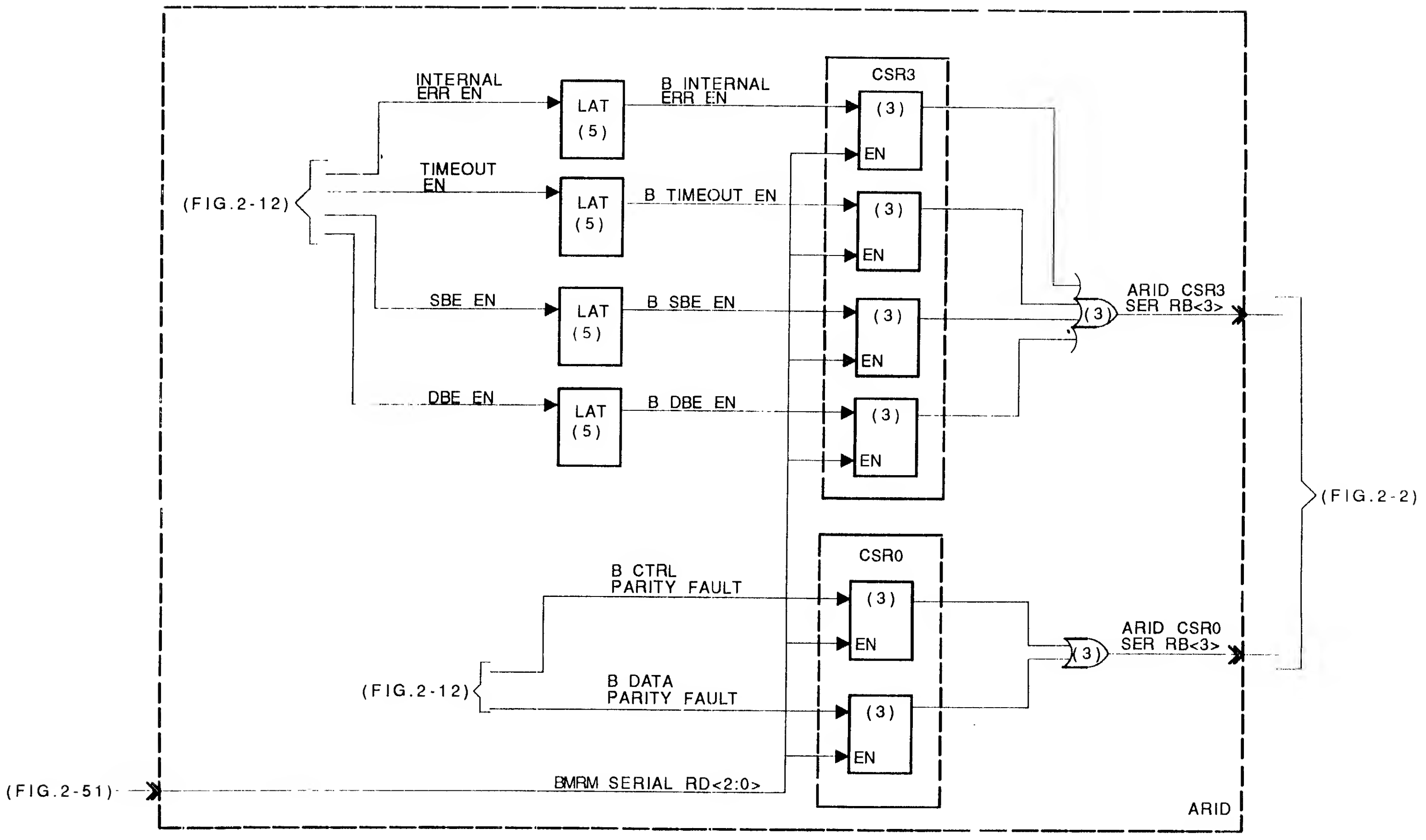
AMRM RESET from the MRM asserts RESET B and RESET A and (via an OR gate) CLEAR B and CLEAR A to the ARID MCA.

ADFA UNJAM from the NMI asserts only CLEAR B and CLEAR A.

The A and B clocks are received from the NMI and are distributed throughout the ARID.

* AMRM INT ERROR and AMDP BUSY REQ assert memory-busy as a means of stopping other nexus from sending commands or data to the memory. AMRM INT ERR asserts when the MRM has detected an internal error. AMDP BUSY REQ asserts when the MDB is full and cannot accept any new data, or when the MDBC has detected a single-bit error.

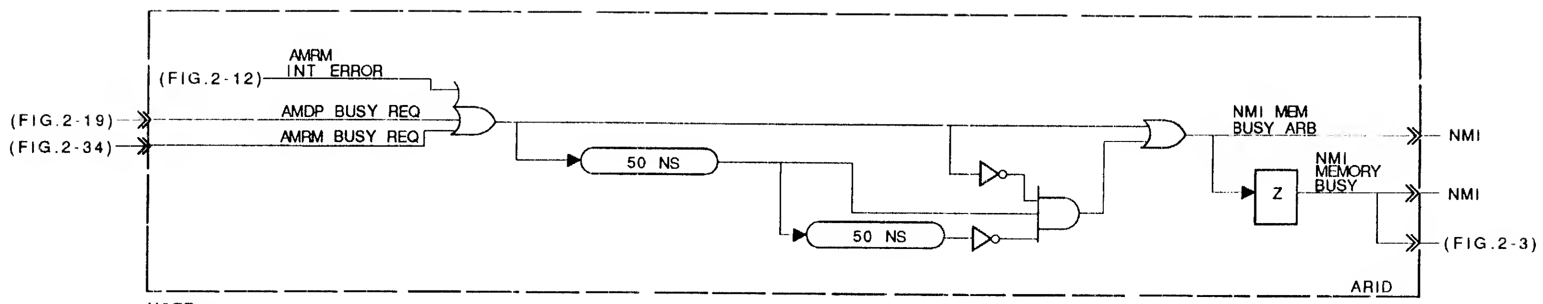
+ To allow time for other nexus to re-arbitrate for the NMI bus.



NOTE:
NUMBER DESIGNATIONS IN PARENTHESES REFER TO ENGINEERING DRAWINGS CONTAINING CORRESPONDING LOGIC.

SCLD-43

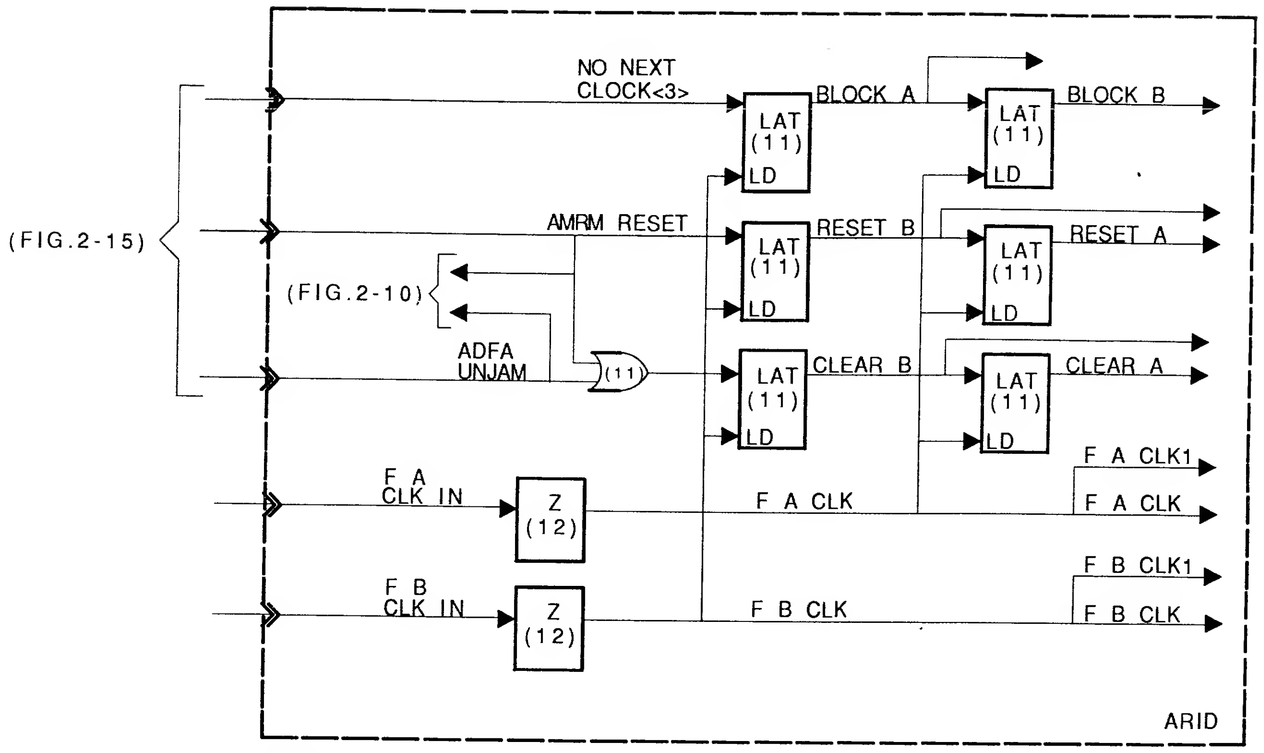
Figure 2-13 CSR Logic



NOTE:
THE LOGIC IN THIS FIGURE IS CONTAINED ON SHEET 8 OF THE ENGINEERING DRAWINGS.

SCLD 44

Figure 2-14 Memory Busy Logic



NOTE:
NUMBER DESIGNATIONS IN PARENTHESES
REFER TO ENGINEERING DRAWINGS
CONTAINING CORRESPONDING LOGIC.

SCLD-47

Figure 2-15 Clock, Reset, and Unjam Logic

2.5 MDP OVERVIEW (Figure 2-16)

The MDP is comprised of the following:

- MDB = memory data buffer: a multi-port RAM (four ports) with sixteen, 36-bit locations. Eight locations are reserved for memory addresses and eight locations are reserved for write data.
- MDBC = memory data buffer control: an MCA that selects the location in the MDB where the write data is placed, and selects the write data that is output from the MDB.
- DCHK = data check: an MCA that performs an ECC check of read data and generates correction bits for single-bit errors.

2.5.1 MDB Address In

In the command/address cycle, the memory address (ADFA DATA ADDR<31:0>) and the address parity bits (ADFA GEN PARITY<3:0>) are received from the DFA and applied to the W write port of the MDB. BMRM ADDR IN SEL<2:0> from the MRM is mux selected as the address input to the MDB where it places the memory address into the address section of the MDB. AMRM INPUT WRT EN<3:0> from the MRM loads the memory address into the selected location.

2.5.2 MDB Address Out

The memory address and the associated parity bits are available at the MDB address read port. The address (BMDP STRD ADDR<25:4>) is converted from ECL to TTL and placed onto the NAB as AMCL ADDR<25:4>. The address parity bits (BMDP STRD APAR<3:0>) are sent to the MRM where they are used to generate a command/address parity bit for the arrays.

The memory address selection is made by selection bits AMRM ADDR OUT SEL<2:0> from the MRM.

2.5.3 MDB Data In

In the write data cycle(s), ADFA DATA ADDR<31:0> and ADFA GEN PARITY<3:0> from the DFA are the write data and the write data parity bits respectively. The load command remains AMRM INPUT WRT EN<3:0> from the MRM. BDFA LD INPUT DATA asserts from the DFA during write-data cycles, causing the MDB address mux to select B DATA IN SEL<2:0> from the MDBC as the MDB location select signal.

B DATA IN SEL<2:0> is generated by the data-in select logic in the MDBC. The MDB data section is divided into two sections (an X buffer and a Y buffer) each having four locations. The data-in select logic selects the first location in the X buffer. For multi-longword writes, the data-in select logic increments through the other X-buffer locations. BDFA LD INPUT DATA asserts from the

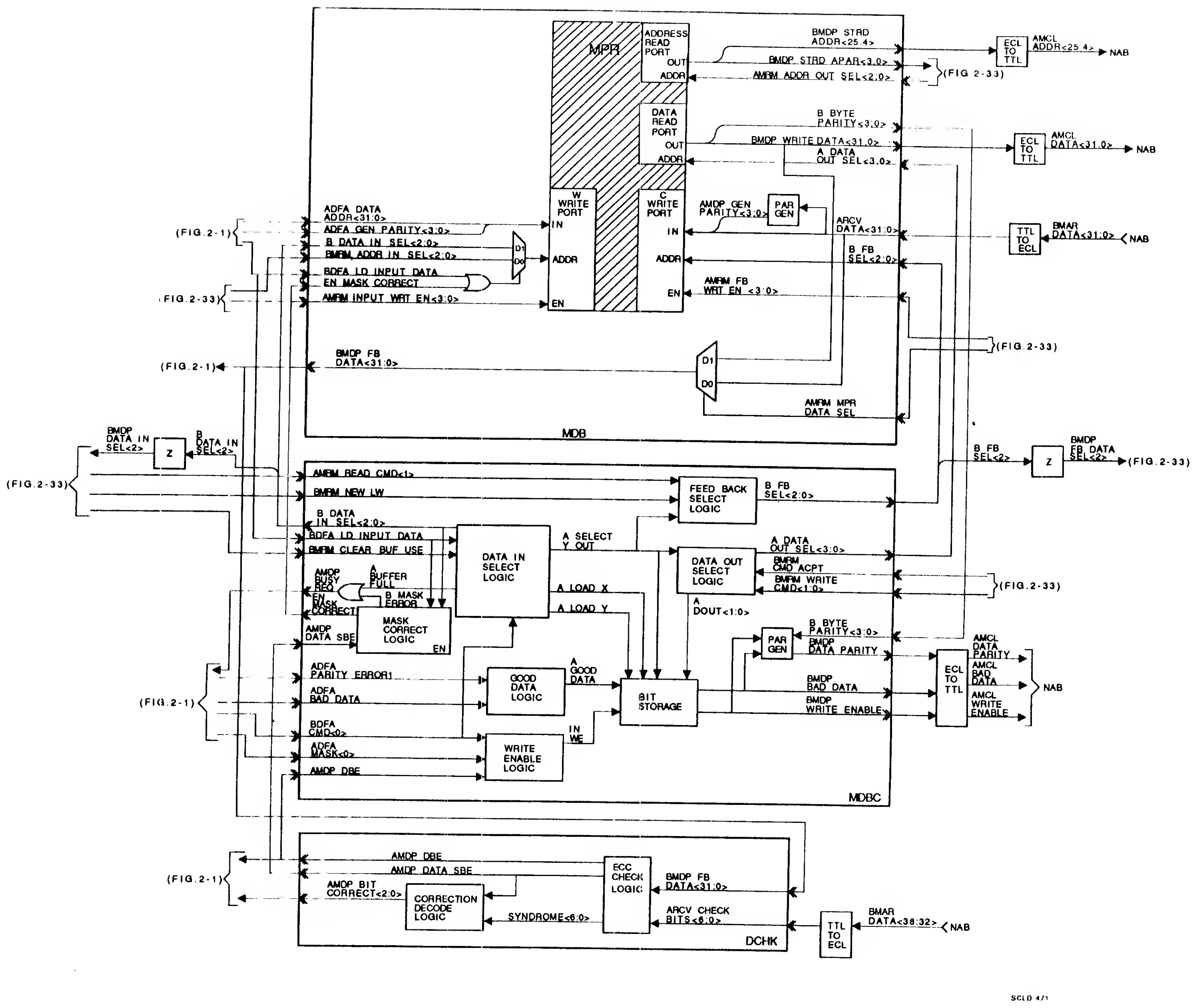


Figure 2-16 Memory Data Path (MDP) Block Diagram

DFA for each write-data cycle causing B DATA IN SEL<2:0> to increment.

When the write data in the X buffer has been transferred to the memory arrays, the MRM asserts BMRM CLEAR BUF USE informing the data-in select logic that the X buffer can be used for the next write command. If the next write command occurs before the data in the X buffer is transferred to the arrays, the data-in select logic places the new write data into the Y buffer. With the X and Y buffers full, the data-in select logic outputs A BUFFER FULL causing AMDP BUSY REQ to assert to the DFA. This places memory-busy on the NMI to halt any more commands to memory.

The data buffer operates on a FIFO basis. Hence, the next assertion of BMRM CLEAR BUF USE indicates that the X data has been transferred to the arrays. The data-in select logic then asserts A SELECT Y OUT to the data-out select logic specifying that the Y buffer is to be unloaded next.

2.5.4 MDB Data Out

The write data and the associated parity bits are available at the MDB data-read port. The write data (BMDP WRITE DATA<31:0>) is converted from ECL to TTL and placed onto the NAB as AMCL DATA<31:0>. The write-data parity bits (B BYTE PARITY<3:0>) are sent to a parity generator in the MDBC where they are used to generate a data parity bit for the arrays.

The data selected for output is determined by A DATA OUT SEL<3:0> from the data-out selct logic in the MDBC. BMRM CMD ACPT from the MRM informs the data-out select logic that the write command has been accepted by the arrays. The data-out select logic selects the first location in the X buffer. For multi-longword writes, BMRM WRITE CMD<1> from the MRM specifies the following longwords as being "next," causing the data-out select logic to increment through the other X locations.

The data-out select logic always selects the MDB X buffer unless A SELECT Y OUT is asserted by the data-in select logic, in which case the Y buffer is selected.

2.5.5 Write-Enable and Bad-Data Bits

Write-enable and bad-data bits are generated in the MDBC and supplied to the arrays with each write-data longword. The bits are generated in good-data logic and write-enable logic, and stored in an eight location bit-storage area. The storage area is divided into a four-location X section and a four-location Y section similar to the MDB.

The good-data logic receives ADFA PARITY ERROR1 and ADFA BAD DATA from the DFA. ADFA PARITY ERROR1 indicates an NMI parity error was detected in the DFA. ADFA BAD DATA indicates a write command sequence error was detected in the DFA. Both ADFA PARITY ERROR1 and ADFA BAD DATA must be false for the good-data logic to assert A GOOD DATA to the bit-storage area.

The write-enable logic receives BDFA CMD<0> and ADFA MASK<0> from the DFA, and AMDP DBE from the DCHK. BDFA CMD<0> informs the write-enable logic if this is a masked operation (see Table 2-1).

- If this is a masked operation (BDFA CMD<0> = 1), AMDP DBE from the DCHK should be false during the read portion of the operation. AN asserted AMDP DBE indicates that the longword read for the masked-write operation, contained an uncorrectable error and the associated longword will not be written. If this is not a masked operation (BDFA CMD<0> = 0), ADFA MASK<0> must be a 1 if the associated longword is to be written (see sort-of-write operation; Section 1.4.3).

The A GOOD DATA and IN WE bits are written into the bit-storage area. The data-in select logic asserts A LOAD X if the X buffer in the MDB is being loaded with write data. A LOAD X places the A GOOD DATA and IN WE bits into the X section of the bit-storage area. For multi-longword operations, A LOAD X is held asserted and the next three pairs of bits are placed into the other three X locations of the bit-storage.

When the data-in select logic is loading data into the Y buffer in the MDB, it asserts A LOAD Y to the bit-storage area which loads the A GOOD DATA and IN WE bits into the Y section of the bit-storage. For multi-longword operations, A LOAD Y is held asserted and the next three pairs of bits are placed into the other three Y locations of the bit-storage area.

The two bits output the storage area as BMDP BAD DATA (inverse of A GOOD DATA) and BMDP WRITE ENABLE which then become AMCL BAD DATA and AMCL WRITE ENABLE respectively on the NAB. The bits are selected for output as the data-out select logic selects the corresponding data longwords from the MDB. The bit-storage will always output from the X section except when A SELECT Y OUT is asserted. A DOUT<1:0> increments the bit-storage output locations for multi-longword operations.

2.5.6 Data Parity

The data parity bits (B BYTE PARITY<3:0>) from the MDB are applied to a parity generator in the MDBC along with the bad-data and write-enable bits. A composite parity bit is generated (BMDP DATA PARITY) which is applied to the NAB as AMCL DATA PARITY. Note that the data parity bit sent over to the arrays represents data parity on the data longword and on the bad-data and write-enable bits.

2.5.7 Data Read Operation

Read data read from the array boards (BMAR DATA<31:0>) is converted from TTL to ECL to become ARCV DATA<31:0> and then applied to a mux in the MDB. The mux outputs the read data to the DFA as BMDP FB DATA<31:0>. BMDP FB DATA<31:0> is also applied to ECC check logic in the DCHK.

Seven ECC check bits (BMAR DATA<38:32>) are received from the array board along with each data longword, converted from TTL to ECL, and then applied to the DCHK ECC check logic as ARCV CHECK BITS<6:0>. The ECC check logic re-generates the ECC check bits from the data longword and compares the re-generated bits to the received check bits to determine if there is a bit-error.

If a single-bit error is detected, the check-logic asserts AMDP DATA SBE and a syndrome code (SYNDROME<6:0>) to correction decode logic. AMDP DATA SBE enables the correction decode logic which decodes the syndrome code to generate correction bits AMDP BIT CORRECT<2:0>. The correction bits are sent to the DAD in the DFA where they correct the bit-error in the data longword.

If a double-bit error is detected, the check-logic asserts AMDP DBE to the DFA where it generates a "read bad data" function code and an NMI interrupt.

2.5.8 Masked Write Operation

In a masked write operation:

- Write data is loaded into the MDB in the normal manner.
- Read data is used to overwrite the write-data.
- The modified write data is transferred from the MDB to the arrays in the normal manner.

The read data overwrites the MDB data via the C write port. The read data (ARCV DATA<31:0>) is applied to the C write port along with parity bits AMDP GEN PARITY<3:0> generated from the read data longword. The read data is byte-enabled to overwrite the MDB data by AMRM FB WRT EN<3:0> from the MRM.

The read data is ECC checked as in the case of a normal read. If a single-bit error is detected AMDP DATA SBE and SYNDROME<6:0> function to provide correction bits (AMDP BIT CORRECT<2:0>) to the DAD in the DFA where the read data is corrected.

AMDP DATA SBE is applied to mask-correction logic in the MDBC. The mask-correction logic is enabled by BMRM NEW LW (a longword of read data has been received) and a negated AMRM READ CMD<1> (the read data is part of a masked write; see Table 2-6).

Table 2-6 Read Command Code

AMRM READ CMD <1 0>	Command *
0 0	Next longword to MDP (masked write)
0 1	First longword to MDP (masked write)
1 0	Next longword to DFA
1 1	First longword to DFA

* Valid when used with BMRM NEW LW.

The mask-correction logic responds to the assertion of AMDP DATA SBE by:

- Asserting B MASK ERROR which causes the assertion of AMDP BUSY REQ to the DFA. In the DFA, AMDP BUSY REQ causes the ARID to place memory-busy on the NMI to halt NMI traffic, and the FUNK to generate ADFA NMI DEAD. ADFA NMI DEAD switches the data path in the DAD to allow the corrected read data to be wrapped around and returned to the MDP where it overwrites the data in the MDP via the W write port.

- Asserting EN MASK CORRECT to the MDB where it switches the W port ADDR input mux to use B DATA IN SEL<2:0> to load the corrected read data into the proper location. The data-in select logic was informed of a masked write operation by BDFA CMD<0> and retained the MDB select address for use when the corrected read data was returned to the MDB. The corrected read data overwrites the MDB data according to the byte-enable signal AMRM INPUT WRT EN<3:0> from the MRM.

If a double-bit error was detected by the DCHK, AMDP DBE is asserted to the write-enable logic in the MDBC which negates the write-enable bit associated with the modified write-data longword. The longword is sent to the array module but is not written into the array.

2.5.9 CSR Reads

When a CSR read operation is executing, the MRM asserts AMRM MPR DATA SEL to the MDB to cause the data-path mux to select write data from the MDB data-read port rather than read data from the arrays. The MDB write data is fed back to the DFA where it is combined with other CSR data and placed onto the NMI.

2.6 MEMORY DATA BUFFER (MDB) (Figure 2-17)

The MDB is a multi-port RAM having two read ports and two write ports. The read ports are designated as the address read port and

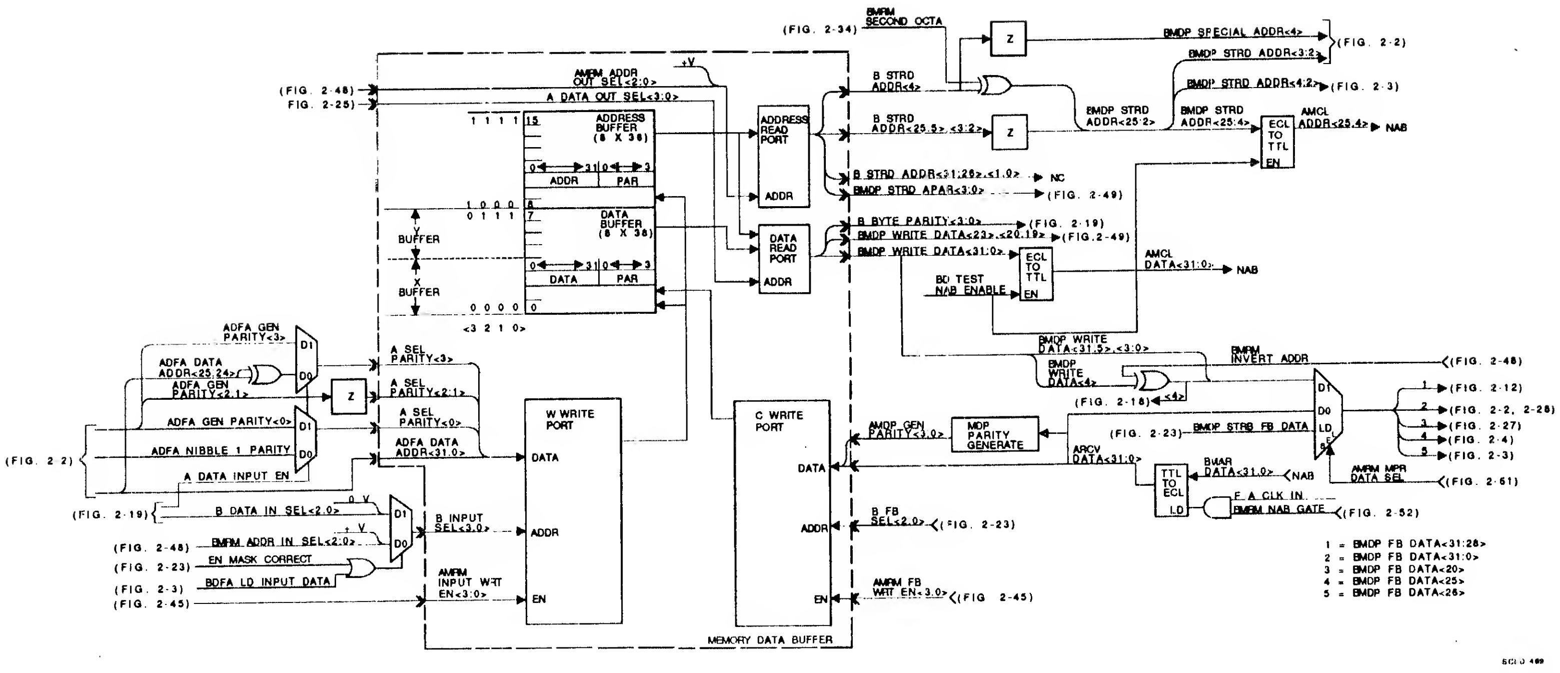


Figure 2-17 Memory Data Buffer (MDB) Block Diagram

the data read port. The write ports are designated as the W write port and the C write port.

The buffer storage area is 16 x 36 with 16 addressable 36-bit locations*. A 36-bit location consists of 32 bits of write data (or a 32-bit address) and four associated parity bits. The lower half of the storage area (locations 0 through 7) is the data buffer where the write data is stored prior to transmission to the MAR4 arrays. The upper half of the storage area (locations 8 through 15) is the address buffer where the command addresses (read and write) are stored prior to transmission to the MAR4 arrays. One of the address locations is used to store an error page address for use in error logging.

The MDB is made up of four identical 16 x 9 RAMs each processing a byte of data and one parity bit. The four RAMs have been combined in Figure 2-17 to present an overall view of MDB operation.

2.6.1 Address Read Port

The address read port outputs command addresses (and parity) from the address buffer. The four parity bits output the port as BMDP STRD APAR<3:0> and are sent to the MRM where a composite parity bit is generated for the MAR4 board.

The command address outputs the port as B STRD ADDR<31:0>. Address bits B STRD ADDR<31:26> and B STRD ADDR<1:0> are not used. B STRD ADDR<25:2> becomes BMDP STRD ADDR<25:2>. BMDP STRD ADDR<4:2> is sent to the FUNK to specify selection of a CSR. BMDP STRD ADDR<3:2> is sent to the DAD where it also specifies a CSR. BMDP STRD ADDR<25:4> is translated from ECL to TTL and becomes AMCL ADDR<25:4>. AMCL ADDR<25:4> is output to the NAB bus as the address for the MAR4 memory arrays.

The ECL to TTL translator is enabled by BD TEST NAB ENABLE from the NAB backplane. BD TEST NAB ENABLE is used for maintenance testing. It is always asserted during normal operation.

Note that address bit B STRD ADDR<4> is XORED with BMRM SECOND OCTA from the MRM. This is used for outputting wrapped hex read addresses. If a hexword-read is being executed, the same address is used from the address read port to read both octawords. When the second octaword is being addressed, the MRM asserts BMRM SECOND OCTA which flips bit 4 of the address. This addresses the next octaword location (the two octaword reads of a hexword read are to contiguous locations).

In addition, B STRD ADDR<4> becomes BMDP SPECIAL ADDR<4> and is applied to the DAD as a third stored address bit for CSR

* The RAM storage area is actually 32 x 36 but only half of the area is used.

selection. It is sent to the DAD before being XORED with BMRM SECOND OCTA because CSR selection requires the address as stored in the MDB.

The addresses to be outputted through the address read port are selected by a four-bit select signal at the port ADDR input. The most significant select bit is a 1 (+V) so only the address buffer locations are selected. The other three select bits (AMRM ADDR OUT SEL<2:0>) select the address buffer location. The three select bits are received from the MRM, hence the MRM controls which addresses output through the address read port.

2.6.2 Data Read Port

In normal operation (not reading CSR1), the output of the data read port is write data (and parity bits) from the data buffer. The four parity bits output the port as B BYTE PARITY<3:0> and are sent to the MDBC where a composite parity bit is generated for the MAR4.

The write data outputs the port as BMDP WRITE DATA<31:0>. Write data bits BMDP WRITE DATA<23> and BMDP WRITE DATA<20:19> are sent to the MRM.

BMDP WRITE DATA<31:0> is translated from ECL to TTL and then applied to the NAB bus as AMCL DATA<31:0>.

The data to be outputted through the data read port is selected by a four-bit select signal at the port ADDR input. The four select bits (A DATA OUT SEL<3:0>) can address all sixteen locations of the memory buffer. For normal operation, the most significant select bit is 0 and the port outputs write data from the data buffer.

The most significant select bit is switched to a 1 when the CSR1 address in the address buffer is to be read. The CSR1 address outputs the address buffer through the data read port and is then fed back to the DAD, via a mux, as BMDP FB DATA<31:0>.

The four select bits for the data read port are received from the MDBC which controls what data outputs from the data buffer. However when the CSR1 address is to be selected from the address buffer, the MRM provides the select bits (via the MDBC).

2.6.3 W Write Port

The input to the memory buffer through the W write port is the data/address from the DAD (ADFA DATA ADDR<31:0>) and its associated parity bits (A SEL PARITY<3:0>).

Parity bits A SEL PARITY<2:1> are derived from ADFA GEN PARITY<2:1>. ADFA GEN PARITY<2:1> is generated in the DAD on bytes two and three of the data/address. Parity bits A SEL PARITY<3> and

A SEL PARITY<0> are input via muxes which select according to whether the input to the W port is an address or write data.

If the port input is an address, A DATA INPUT EN from the MDBC is false and the muxes select their D0 inputs. In this case, A SEL PARITY<0> is ADFA NIBBLE 1 PARITY which is parity generated in the DAD on address bits <7:4>. A SEL PARITY<3> is a parity bit generated (XORed) on address bits <25:24>. This alters the parity bits so that they reflect parity on the 22 address bits (<25:4>) used to address the MAR4 array boards.

If the port input is write data, A DATA INPUT EN is asserted and the muxes select the parity bit generated in the DAD on data byte 3 (ADFA GEN PARITY<3>) and the parity bit generated on data byte 0 (ADFA GEN PARITY<0>). This provides four parity bits for the full 32-bit data longword.

The location where the input data is to be written is selected by a four-bit select input (B INPUT SEL<3:0>) at the W port ADDR input. The four select bits can address all sixteen locations of the memory buffer. The select bits are obtained through a mux which selects according to whether the input to the W port is an address or write data.

If the port input is an address, the mux selects its D0 input. The most significant bit of the D0 input is a 1 (+V) which selects the address buffer. The other three bits (BMRM ADDR IN SEL<2:0>) select which of the eight locations of the address buffer is to be written. The select bits are obtained from the MRM.

If the port input is write data, the mux is switched to select the D1 input. The most significant bit of the D1 input is 0 which selects the data buffer. The other three bits (B DATA IN SEL<2:0>) select which of the eight locations in the data buffer is to be written. The select bits are obtained from the MDBC.

Either of two signals can switch the mux to select its D1 input. One is BDFA LD INPUT DATA obtained from the FUNK, during a write-data cycle, when write data is being processed. The other is EN MASK CORRECT asserted by the MDBC when masked read data was corrected by the DAD and is being written into the MDB.

The W write port is byte enabled by AMRM INPUT WRT EN<3:0> from the MRM. With the enable input being four bits, the MRM controls the writing process by allowing all the bytes to be written for unmasked operations, and selected bytes to be written for the read-data correction cycle of a masked write operation.

2.6.4 C Write Port

Data is input to the memory buffer through the C write port during the read portion of a masked write operation. The data consists of the read data longword (ARCV DATA<31:0>) from the memory array NAB bus, and its associated parity bits (AMDP GEN PARITY<3:0>). The read data from the NAB (BMAR DATA<31:0>) is translated from TTL to ECL to become ARCV DATA<31:0>. The TTL to ECL translator is enabled by BMRM NAB GATE from the MRM.

The four parity bits are generated on the read data in the MDP parity-generate logic.

The location where the read data is to be written is selected by a three-bit select input (B FB SEL<2:0>) at the C port ADDR input. The three select bits can address only the eight locations of the data buffer. The select bits are obtained from the MDBC which places the read data in the data buffer so as to overwrite the masked write data.

The C write port is byte enabled by AMRM FB WRT EN<3:0> from the MRM. With the enable input being four bits, the MRM controls the writing process by overwriting specific bytes according to the associated mask field.

2.6.5 CSR Logic

Figure 2-18 illustrates CSR data supplied from the MDB area to the CSR logic in the DAD. BMRM SERIAL RD<2:0> is supplied from the MRM to the select inputs of a mux and serial-read logic. BMRM SERIAL RD<2:0> is incremented to cause the mux and the serial-read logic to convert multi-bit inputs into one-bit serial outputs. In eight cycles, all the inputs have been converted to serial outputs.

The CSR1 data (BMDP CSR1 SER RB<0>) is a one-bit output obtained from a mux, and carries the primary array board number (ADFA PRM BNUM<2:0>), the alternate array board number (ADFA ALT BNUM<2:0>), and the memory-address bit (ADFA MEM ADDR) from the decode-RAM in the DAD, and write-data bit <4> from the MDB.

The CSR3 data (BMDP CSR3 SER RD<2:0>) are three bits obtained from serial-read logic, that indicate the array-board size. A three-bit code from each array board specifies the array board size. The 24 bits from all the boards (ARRAY SIZE<23:0>) are output to CSR3 in three, eight-bit serial segments.

The CSR0 data (BMDP CSR0 SER RD<2:0>) consist of:

- An eight-bit revision code
- 0 volts
- The most-significant bit of the serial-read select bits (BMRM SERIAL RD<2>)

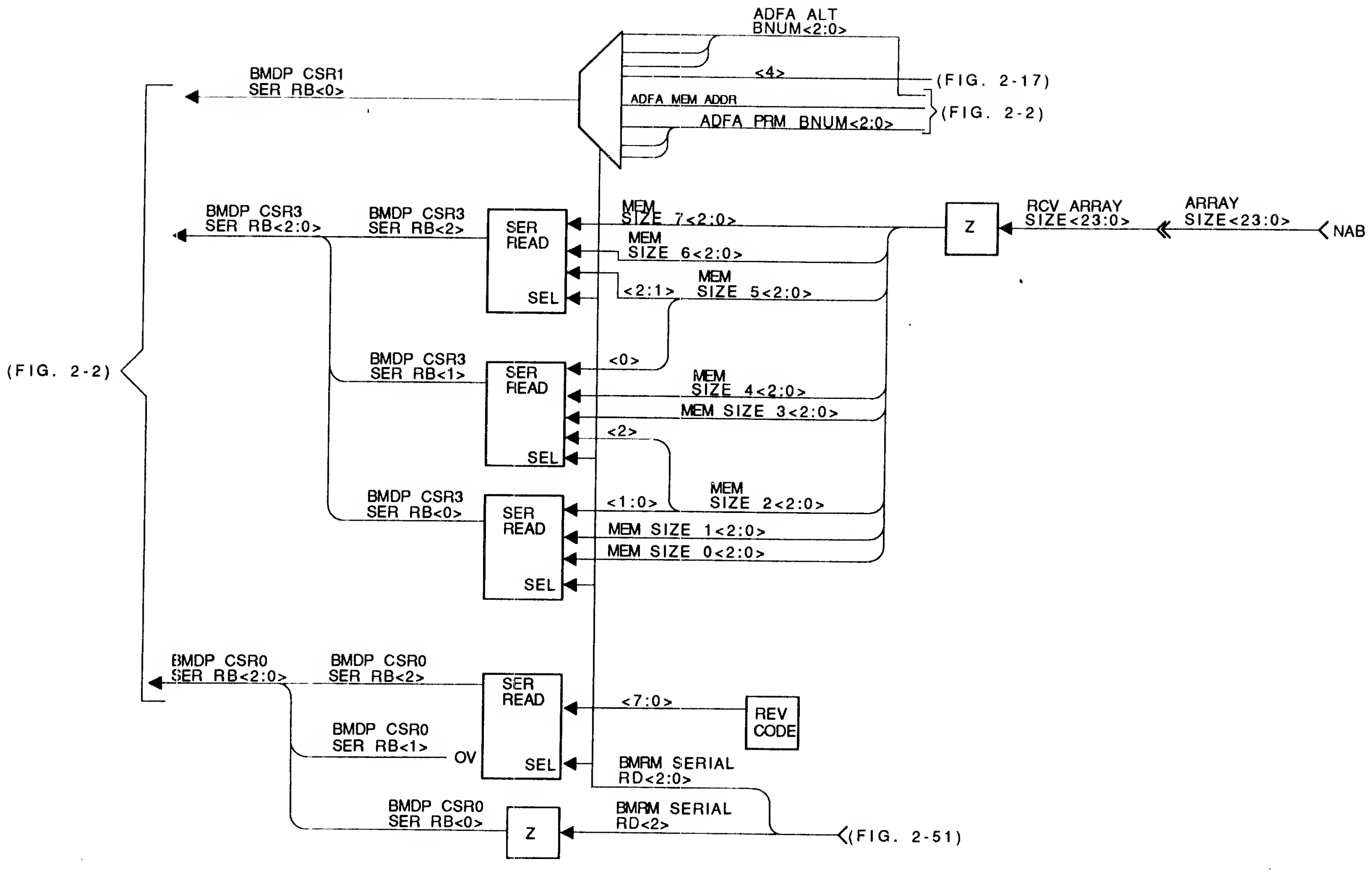


Figure 2-18 CSR Logic

2.7 MEMORY DATA BUFFER CONTROL (MDBC) MCA

The main function of the MDBC is to control operation of the MDB. MDB operations controlled by the MDBC are:

- Selecting the location in the data buffer for the write data that is loaded via the W write port. The select signal that points to the write data is B DATA IN SEL<2:0>. Enabling of the MDB to write the data is done by the MRM.

- Selecting the location in the data buffer for the feedback data that is read from the MAR4 and loaded into the data buffer via the C write port. The select signal that points to the feedback data is B FB SEL<2:0>. Enabling of the MDB to write the feedback data is done by the MRM.

- Selecting the write data in the data buffer and the CSR1 data in the address buffer that is to be unloaded via the data read port. The select signal that selects the write data or the CSR1 data is A DATA OUT SEL<3:0>. The MDBC has complete control of the unloading process as the MDB does not require an enabling signal to unload data.

Other functions of the MDBC are to:

- Store up to eight pairs of bad-data and write-enable bits associated with the data stored in the data buffer, and unload them as their associated data longwords are unloaded from the MDB.

- Generate a data parity bit for the write data unloaded from the MDB.

- Generate a busy request for the FUNK when the data buffer is full.

- Notify the MRM when a masked write operation is completed.

- Generate a delayed internal-error signal for the MRM.

- Generate a write strobe for the decode ram.

The data buffer portion of the MDB is divided into halves; an X buffer consisting of four longwords (locations 0 through 3) and a four-longword Y buffer (locations 4 through 7). Each buffer holds the write data associated with a write command. The MDBC always tries to place the write data into the X buffer. The only time the MDBC will place data into the Y buffer is when the X buffer already has data.

The data from only one write command is placed into a buffer. If a longword of write data is in the X buffer and another longword write command occurs, the new longword is placed into the Y buffer even though there are three empty locations in the X buffer.

In the following discussions of normal and masked writes, octaword writes are described. The only way the MDBC knows the size of a transfer is by the duration of the input load command (BDFA LD INPUT DATA or AMRM READ CMD<1>) or the output unload command (BMRM WRITE CMD <1:0>). A longword or quadword transfer is only the first cycle or the first two cycles of an octaword transfer. Hence, longword and quadword writes are implicitly included in the description of octaword writes.

The block diagram of the MDBC MCA is contained in four illustrations (Figures 2-19, 2-23, 2-25, and 2-27). The four illustrations divide the MDBC according to function. All of the MDBC I/O signals are shown in the illustrations.

2.7.1 Loading Data into the MDB

Loading write data into the MDB is done for normal writes and for masked writes. During a masked write, feedback data from the NAB is also loaded into the MDB to overwrite the data already there. Normal writes are considered first.

2.7.1.1 Normal Octaword Write With X and Y Buffers Empty -- Figure 2-19 is a block diagram of the MDBC data-in selection function. The major areas shown in Figure 2-19 are defined below.

- Full Logic -- indicates that data is stored in the X or Y buffer by asserting B X VALID or B Y VALID respectively.
- Input Load CMD Detect -- senses load commands and the status of B X VALID. Indicates whether the new data is for the X or Y buffer by asserting B NEW X or B NEW Y respectively.
- Input Load Counter -- provides a two-bit select output (B IN COUNT<1:0>) used as the two least significant bits of the B DATA IN SEL<2:0> input selection signal. Also used as a select input for the X & Y bit-storage. When enabled, the counter is incremented each bus cycle.
- X & Y bit-Storage -- an eight-deep storage buffer which stores the good data and write-enable bits associated with the data longword(s) stored in the X & Y buffers.

When write data is to be loaded into the MDB, the FUNK asserts BDFA LD INPUT DATA into the MDBC and holds it asserted for the entire transfer (one cycle for a longword transfer; four cycles for an octaword transfer). The assertion of BDFA LD INPUT DATA does the following:

- Asserts A DATA INPUT EN to the MDB to select the data parity (rather than the address parity) as an input to the W write port.

- Latches B LOAD Y IN which is applied to the selection output mux. B LOAD Y IN is 0 due to B X VALID being false (the X buffer is empty).

- Enables the input-load counter. Once enabled, the counter output (B IN COUNT<1:0>) is incremented from a count of 00 by F A CLK and F B CLK.

- Switches the selection output mux to its D1 input thereby outputting B LOAD Y IN (presently 0) as the most significant bit of B DATA IN SEL<2:0>, and B IN COUNT<1:0> as the two least significant bits. This starts the B DATA IN SEL<2:0> output at 000 which is then incremented to 011 by the input load counter to select the four locations of the X buffer.

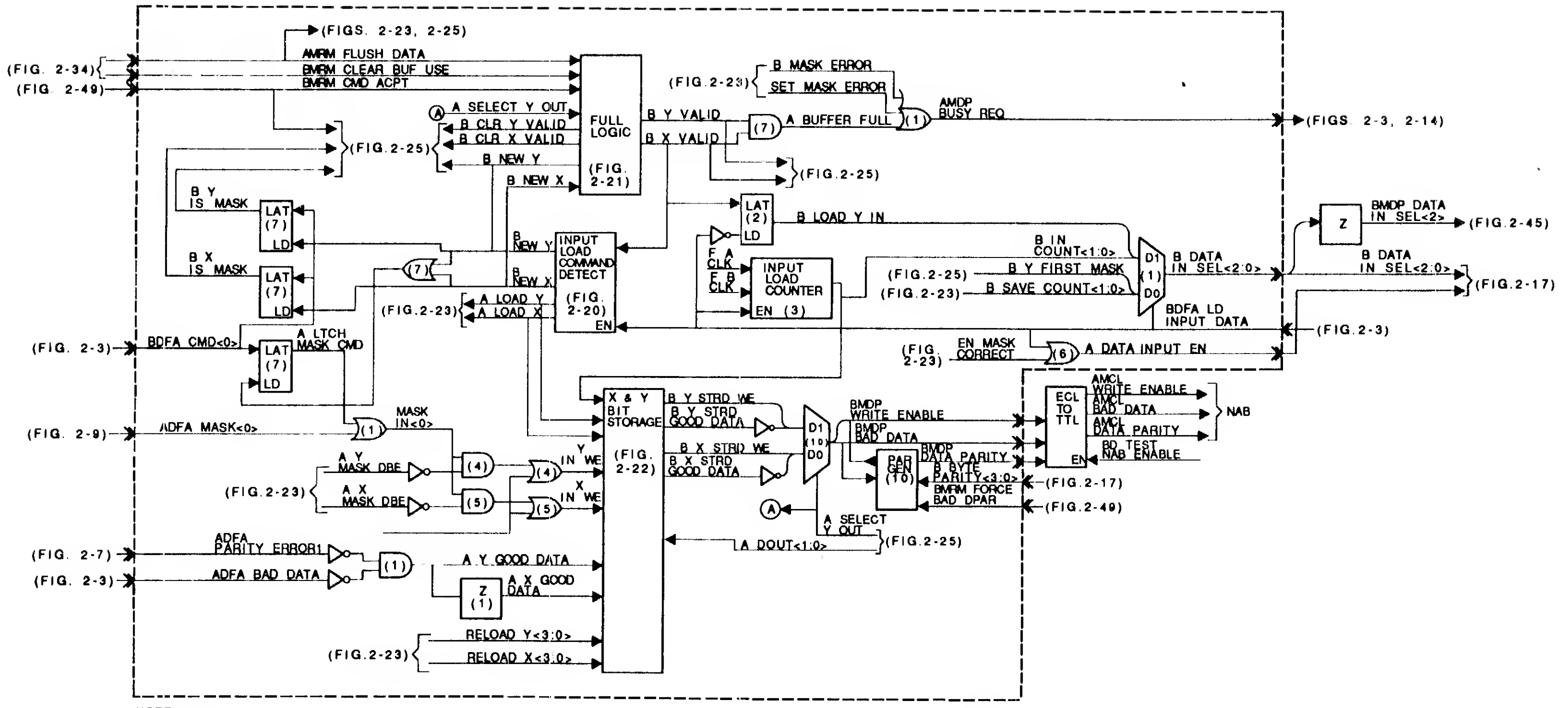
- Enables the input-load-CMD detect logic (Figure 2-20) which asserts B NEW X indicating that new data is going into the X buffer. B X VALID is false as the X buffer is empty. B NEW X asserts A LOAD X via a latch. BDFA LD INPUT DATA is delayed one cycle and then negates B NEW X but latches A LOAD X for the duration of the transfer.

B NEW X is applied to the full logic (Figure 2-21) where it asserts B X VALID indicating that data is in the X buffer. B X VALID latches itself until cleared by B CLR X VALID.

B CLR X VALID is asserted after the X data has been unloaded from the MDB. After the data has been unloaded, the MRM asserts BMRM CLR BUF USE while BMRM CMD ACPT is true. A SELECT Y OUT asserts if the Y buffer is selected for an unload (see Section 2.7.3). As the X buffer was selected for the unload, A SELECT Y OUT is false and B CLR X VALID asserts. B CLR X VALID is also asserted by AMRM FLUSH DATA as part of a reset function.

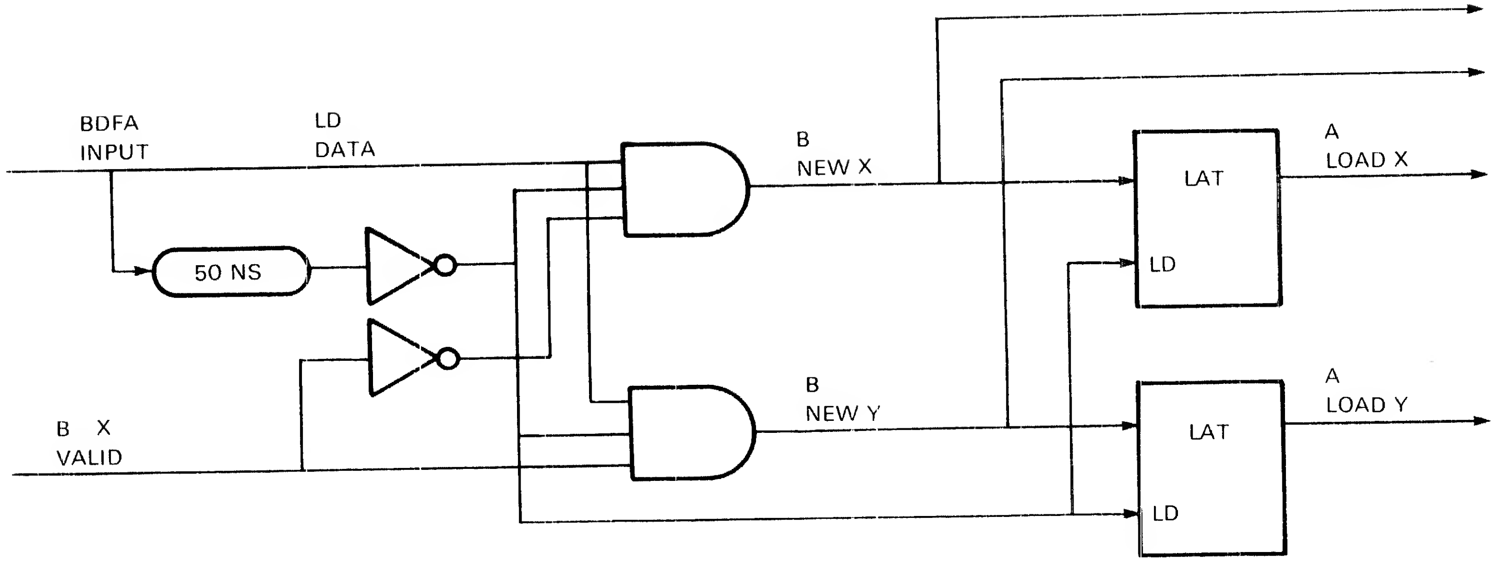
In addition, B NEW X checks for a masked operation by loading BDFA CMD<0> into a latch. If this were a masked operation, BDFA CMD<0> would be true and B X IS MASK would be asserted and latched for the duration of the transfer. B X IS MASK is sent to the Y-out-select logic to indicate that the data going into the X buffer is masked data.

A LOAD X is applied to the X & Y bit-storage block (Figure 2-22) where it enables the B IN COUNT<1:0> bits to select the location within the X bit-storage block where the good-data and the write-enable bits are to be stored. B IN COUNT<1:0> is applied to a decoder having four outputs. As B IN COUNT<1:0> is incremented, the four decoder outputs are asserted in sequence. This causes the four LOAD X<3:0> select signals to be asserted in sequence to select the four storage locations within the X bit-storage block.



NOTE:
NUMBER DESIGNATIONS IN PARENTHESES
REFER TO ENGINEERING DRAWINGS
CONTAINING CORRESPONDING LOGIC.

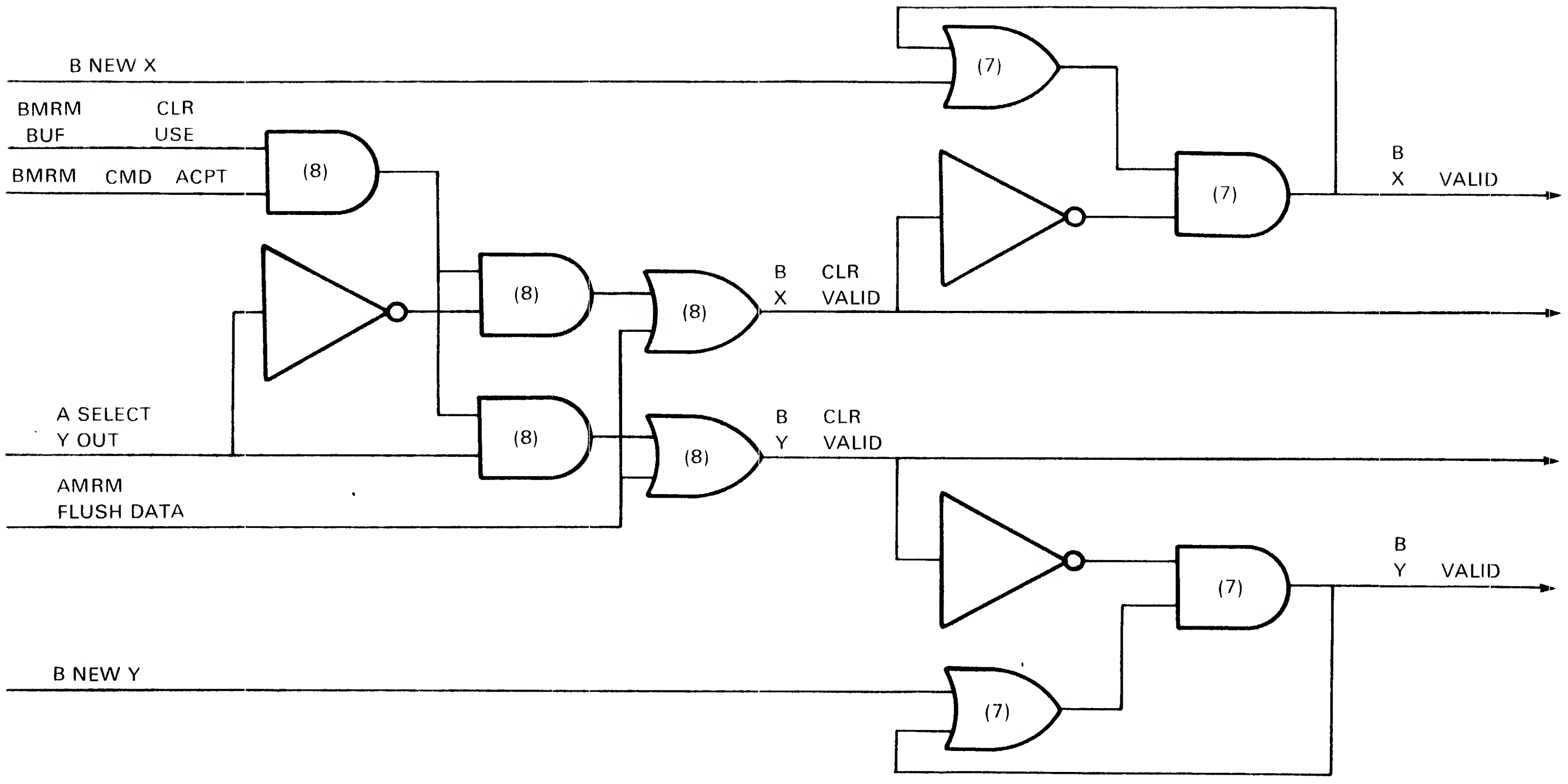
Figure 2-19 MDBC -- MDB Data-In Selection



NOTE:
THE LOGIC IN THIS FIGURE IS CONTAINED ON SHEET 2 OF THE ENGINEERING DRAWINGS.

MKV86-1289

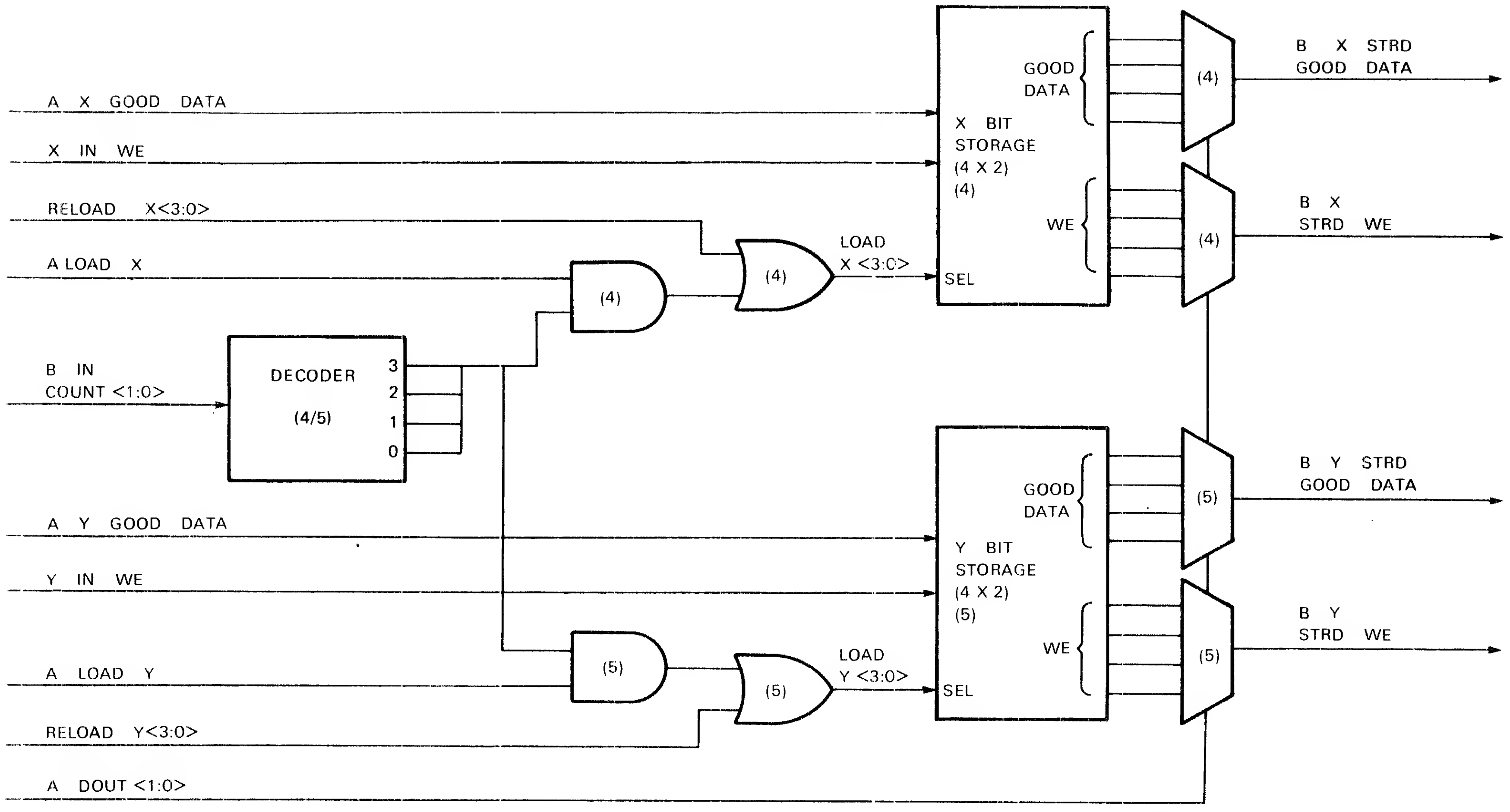
Figure 2-20 Input Load Command Detect Logic



NOTE:
NUMBER DESIGNATIONS IN PARENTHESES REFER TO ENGINEERING DRAWINGS CONTAINING CORRESPONDING LOGIC.

MKV86-1290

Figure 2-21 Full Logic



NOTE:
NUMBER DESIGNATIONS IN PARENTHESES REFER TO ENGINEERING DRAWINGS CONTAINING CORRESPONDING LOGIC.

MKV86-1296

Figure 2-22 X and Y Bit Storage

The good-data bit (A X GOOD DATA) is derived from the parity error bit from the ARID and the bad data bit from the FUNK. If there were no data or control parity errors detected in the ARID, ADFA PARITY ERROR1 is false. If there was no write sequence error detected in the FUNK, ADFA BAD DATA is false. If both error inputs are false, A X GOOD DATA is asserted to the X bit-storage block indicating that the associated data longword being stored in the X buffer is good data.

The write-enable bit (X IN WE) is generated by an OR gate with one of the OR inputs derived from bit <0> of the write mask and the double-bit error bit from the DBE logic. In a normal (non-masked) write operation, bit <0> of the write mask (ADFA MASK<0>) is a 1 if the associated longword is to be written (sort-of-write operation). With ADFA MASK<0> true, MASK IN<0> is also true. In a masked write operation, BDFA CMD<0> is a 1 (see Table 2-1) also causing MASK IN<0> to be true. The double-bit error function is disabled during normal writes (Section 2.7.1), hence A X MASK DBE will be false. With A X MASK DBE false and MASK IN<0> true, X IN WE is asserted to the X bit-storage block indicating that the associated data longword stored in the X buffer is to be written into the memory arrays.

The other OR input that will assert the write-enable bit is ADFA PARITY ERROR1. The condition under which ADFA PARITY ERROR1 asserts X IN WE is during a sort-of-write operation when a given longword is not to be written (ADFA MASK<0> false) and the ARID detected a parity error on the longword*. ADFA PARITY ERROR1 forces the write-enable bit set so the longword will be written for use in error analysis. The longword is written as bad data, because ADFA PARITY ERROR1 will negate the good-data bit.

As the second, third, and fourth longword is loaded into the MDB, B IN COUNT<1:0> (and then B DATA IN SEL<2:0>) is incremented to locate the write data longwords into locations 0 through 3 of the X buffer. B IN COUNT<1:0> also loads the four associated good-data and write-enable bits into the X bit-storage block.

* The existence of a parity error does not force the write-enable bit when a double-bit error is detected because by the time the double-bit error has been detected, ADFA PARITY ERROR1 has negated, hence the double-bit error negates the write-enable bit as it normally would.

After the last data longword has been entered into the X buffer, the Funk negates BDFA LD INPUT DATA. The negation of BDFA LD INPUT DATA prepares the MDBC for the next write (which will be into the Y buffer if the X data has not been unloaded) by:

- Loading B X VALID into a latch thereby asserting B LOAD Y IN. With B LOAD Y IN asserted, the most significant bit of B DATA IN SEL<2:0> becomes a 1 thereby selecting the Y buffer for the next write operation.

- Disabling the input load counter, thereby resetting it to zero to address the first location in the Y buffer (location 4).

- Disabling the input-load-CMD detect logic, thereby negating A LOAD X which disables the X bit-storage block.

- Negating A DATA INPUT EN so that the MDB will select the address parity as an input to the W write port.

2.7.1.2 Normal Octaword Write With Data Already in the X or Y Buffer -- If another write operation is executed with data still in the X buffer, BDFA LD INPUT DATA will assert from the FUNK and do the following:

- Assert A DATA INPUT EN to the MDB to select the data parity (rather than the address parity) as an input to the W write port.

- Latch B LOAD Y IN. B LOAD Y IN is now true due to B X VALID being true (the X buffer contains data). B LOAD Y IN is applied to the selection output mux .

- Enable the input load counter to increment B IN COUNT<1:0>.

- Switch the selection output mux to its D1 input thereby outputting B LOAD Y IN (presently true) as the most significant bit of B DATA IN SEL<2:0>, and B IN COUNT<1:0> as the two least significant bits. This starts the B DATA IN SEL<2:0> output at 100 which is then incremented to 111 by the input load counter to select the four locations of the Y buffer.

- Enable the input-load-CMD logic which asserts B NEW Y indicating that new data is going into the Y buffer (B X VALID true). B NEW Y asserts A LOAD Y via a latch. BDFA LD INPUT DATA is delayed one cycle and then negates B NEW Y but latches A LOAD Y for the duration of the transfer.

B NEW Y is applied to the full logic where it asserts B Y VALID indicating that data is in the Y buffer. B Y VALID latches itself until cleared by B CLR Y VALID. B CLR Y VALID is asserted by the assertion of BMRM CLR BUF USE while BMRM CMD ACPT is true. BMRM CLR BUF USE is asserted after the data in the Y buffer has been unloaded. B CLR Y VALID is also asserted by AMRM FLUSH DATA as part of a reset function.

With both B X VALID and B Y VALID asserted, A BUFFER FULL asserts in turn asserting AMDP BUSY REQ to the FUNK. The busy request sent to the FUNK will cause memory-busy on the NMI indicating that the data buffer is full and cannot accept any more data.

A LOAD Y is applied to the X & Y bit-storage block where it enables the B IN COUNT<1:0> bits to select the location within the Y bit-storage block where the good-data and the write-enable bits are to be stored. As B IN COUNT<1:0> is incremented, the four LOAD Y<3:0> select signals are asserted in sequence to select the four storage locations within the Y bit-storage block.

The good-data bit (A Y GOOD DATA) is derived from the parity error bit from the ARID and the bad data bit from the FUNK as described for the X bit-storage block.

The write-enable bit (Y IN WE) is generated by an OR gate with one of the OR inputs derived from bit <0> of the write mask and the double-bit error bit from the DBE logic, as described for the X bit-storage block. The other OR input is ADFA PARITY ERROR1 which also functions as described for the X bit-storage block.

As the second, third, and fourth longword is loaded into the MDB, B IN COUNT<1:0> (and then B DATA IN SEL<2:0>) is incremented to locate the write data longwords into locations 4 through 7 of the Y buffer. B IN COUNT<1:0> also loads the four associated good-data and write-enable bits into the Y bit-storage block.

After the last data longword has been loaded into the Y buffer, the FUNK negates BDFA LD INPUT DATA which negates A LOAD Y in the input-load-CMD detect logic.

Data must be output from the data buffer (X buffer first) before any more commands (read or write) can be processed by memory.

If data was in the Y buffer (instead of the X buffer) when the write command was received, a similar sequence is executed to load data into the X buffer, but with the following differences:

- B LOAD Y IN is false as B X VALID is false (no data in X buffer). Consequently, B DATA IN SEL<2> is 0 and the X buffer is selected for the new data.

- B NEW X and A LOAD X are asserted. A LOAD X enables the X bit-storage block.

- After the X buffer is loaded and AMDP BUSY REQ asserts, it is the Y buffer that must be unloaded first before anymore commands can be processed.

2.7.1.3 Masked Write With No Errors --

A. General

In a masked write operation, write data is placed into the X (or Y) buffer via the B DATA IN SEL<2:0> select signals as described in Sections 2.7.1.1 and 2.7.1.2. Feedback data read from the arrays is checked for errors in the DCHK. If no errors are detected, the data is applied to the X (or Y) buffer where it overwrites specific bytes as determined by the byte mask in the MRM. The MDBC selects the location(s) within the data buffer that is to receive the feedback data.

Figure 2-23 is a block diagram of the MDBC feedback selection function. The major areas shown in Figure 2-23 are defined below.

- Feedback Counter -- provides a two-bit select output (B FB SEL<1:0>) used as the two least significant bits of the B FB SEL<2:0> feedback input selection signal. When enabled, the counter is incremented each bus cycle.
- DBE Logic -- senses a double-bit error (AMDP DBE) from the DCHK during a masked write operation, and asserts A X MASK DBE (or A Y MASK DBE) to the X (or Y) reload decoder and bit-storage block.
- X Reload -- reloads a negated write-enable bit into the X bit-storage for any longword in which a double-bit error was detected during the feedback-read portion of a masked write to the X buffer.
- Y Reload -- same as the X reload block but for the Y write-enable bit during masked write to the Y buffer.

B. Detailed

The signal that selects the location(s) of the feedback data is B FB SEL<2:0>. The most-significant bit (B FB SEL<2>) is B Y FIRST MASK from the Y-select logic. The Y-select logic asserts B Y FIRST MASK when the feedback data is for the Y buffer. This is discussed in the description of the Y-select logic contained in Section 2.7.3.

The two least significant bits (B FB SEL<1:0>) are obtained from the feedback counter. The counter is cleared by BMRM RESET MASK CTR from the MRM when a masked write operation is initiated. The counter is enabled by A MDP SEL which is derived from BMRM NEW LW and the inverse of AMRM READ CMD<1>. BMRM NEW LW asserts for each new longword of data being transferred. AMRM READ CMD<1> is 0 (negated) when the read data is supplied to the MDB. (Read data being supplied to the MDB is part of a masked write operation; see Table 2-6.) When enabled, the counter is incremented from 00 to 11 by F A CLK and F B CLK.

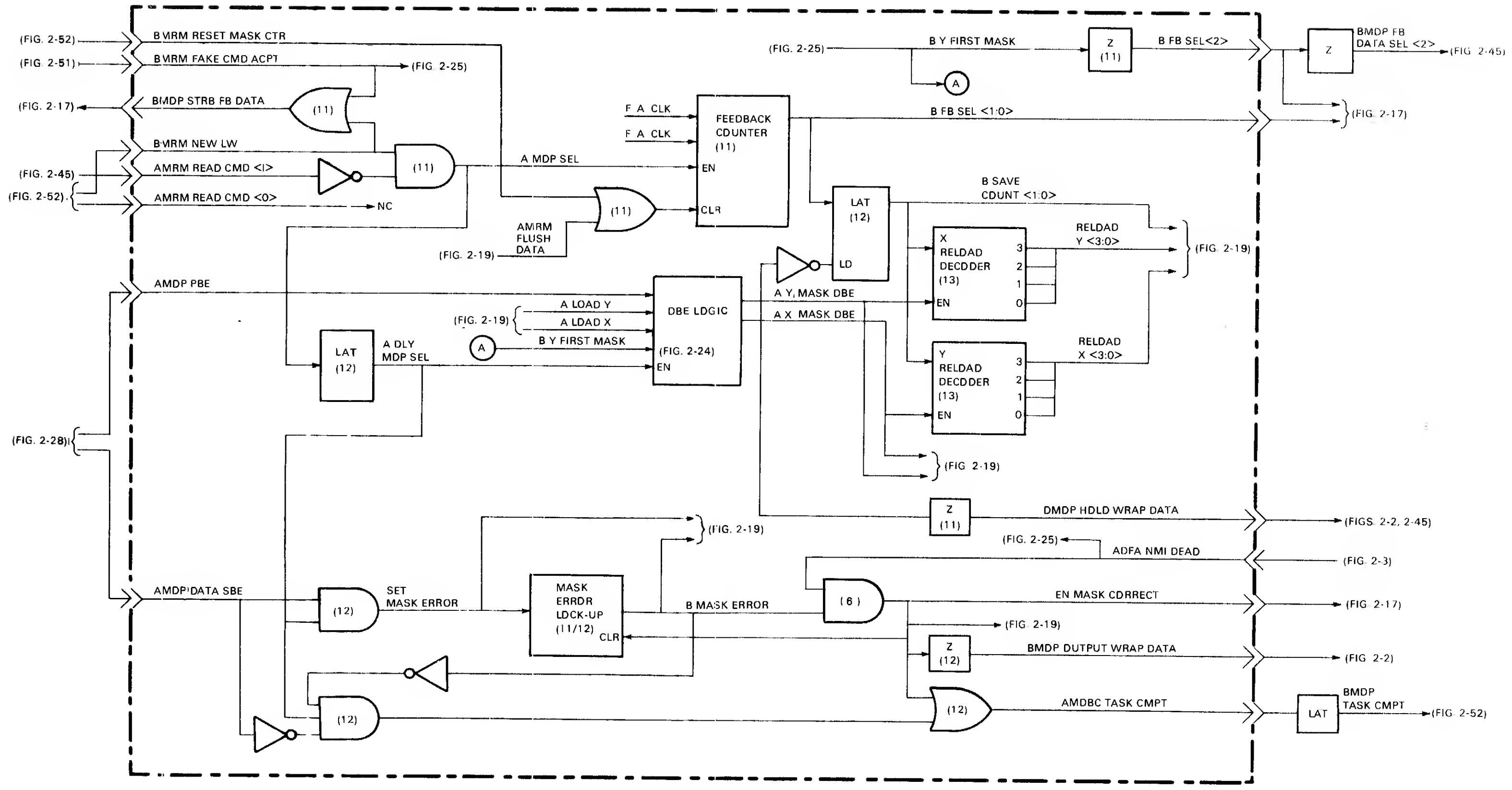


Figure 2-23 MDBC -- MDB Feedback Selection

In addition, BMRM NEW LW asserts BMDP STRB FB DATA to the feedback mux to strobe feedback data to the DAD (Figure 2-17). BMRM FAKE CMD ACPT is used to assert BMDP STRB FB DATA for maintenance testing.

2.7.1.4 Masked Write With a Correctable Error --

A. General

Feedback data from the NAB is checked for errors in the DCHK. If a single-bit error is detected, the location select signal is saved while the data is routed through the DAD for error correction. The corrected data is then input to the data buffer through the W write port where its location in the buffer is determined by the location select signal that was saved.

Refer to Figure 2-23 for the following discussion.

B. Detailed

A MDP SEL is held asserted while the feedback data from the NAB is read back. A MDP SEL asserts A DLY MDP SEL one cycle later (after the DCHK has checked the feedback data). A DLY MDP SEL checks for the presence of a single-bit error (AMDP DATA SBE). If there is none, AMDBC TASK CMPT is asserted (via an OR gate) to indicate that the masked write of the longword has been completed. AMDBC TASK CMPT becomes BMDP TASK CMPT for the MRM.

If a single-bit error is detected, AMDP DATA SBE asserts which inhibits the assertion of AMDBC TASK CMPT. In addition, AMDP DATA SBE asserts SET MASK ERROR which in turn asserts AMDP BUSY REQ to the FUNK (see Figure 2-19). AMDP BUSY REQ causes memory-busy to assert on the NMI to stop traffic to the MCL so that the DAD can return the corrected data to the MDB.

SET MASK ERROR is also applied to a mask-error lock-up block which outputs B MASK ERROR. B MASK ERROR performs the following:

- Holds AMDBC TASK CMPT negated until the mask write is completed.
- Holds AMDP BUSY REQ asserted until the corrected data is returned from the DAD.
- Latches up the location select signal from the feedback counter as B SAVE COUNT<1:0>. B SAVE COUNT<1:0> is the location in the MDB where the corrected data is to be written when it is returned from the DAD.
- Asserts BMDP HOLD WRAP DATA to the DAD where it holds the corrected data until the NMI traffic has stopped as indicated by the assertion of ADFA NMI DEAD from the FUNK.

When ADFA NMI DEAD asserts, EN MASK CORRECT asserts to resume the masked write sequence as follows:

- Switches the MDB W port ADDR mux to select the B DATA IN SEL<2:0> select signal from the MDBC.
- Asserts A DATA INPUT EN (see Figure 2-19) to the MDB W port to select the data parity bits for the DATA input.
- Asserts BMDP OUTPUT WRAP DATA to the DAD to enable the corrected feedback data to flow to the MDB.
- Asserts AMDBC TASK CMPT to the MRM indicating that the mask write is completed.
- Clears the mask-error lock-up logic thereby negating B MASK ERROR. The negation of B MASK ERROR negates the NMI busy request and BMDP HOLD WRAP DATA. The negation of BMDP HOLD WRAP DATA allows the corrected feedback data to wrap around the DAD and return to the MDB.

The latched location signal (B SAVE COUNT<1:0>) is selected as the two least significant bits of the B DATA IN SEL<2:0> select signal while B Y FIRST MASK is selected as the most-significant bit. This writes the corrected data into the proper location in the data buffer.

2.7.1.5 Masked Write With an Uncorrectable Error --

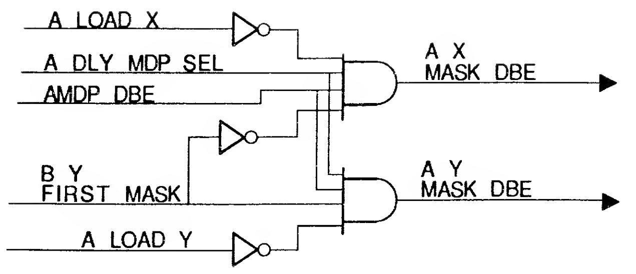
A. General

When an uncorrectable error occurs during a masked write operation, a negated write-enable bit is reloaded into the bit-storage block for the longword containing the uncorrectable error. The write-enable bit has already been stored by the time the double-bit error has been detected, hence the loading of the negated bit is a reload operation. When the erroneous longword is output to the NAB, the associated write-enable bit will be false and the longword will not be written into the memory arrays.

Refer to Figure 2-23 for the following discussion.

B. Detailed

When the DCHK detects an uncorrectable error in the feedback data, it asserts AMDP DBE to the DBE logic (Figure 2-24). A DLY MDP SEL enables the DBE logic which senses the presence of the double-bit error. The DBE logic looks at B Y FIRST MASK to determine which buffer is being loaded. Accordingly, it asserts A X MASK DBE (or A Y MASK DBE) which negates X IN WE (or Y IN WE) to the appropriate bit-storage block. Note that the DBE logic is disabled by A LOAD X (or A LOAD Y) during normal writes to the data buffer.



NOTE:
THE LOGIC IN THIS FIGURE IS CONTAINED ON SHEET 13 OF THE ENGINEERING DRAWINGS.

SCLD-477

Figure 2-24 Double-Bit Error Logic

The location of the erroneous longword exists in the B SAVE COUNT<1:0> code obtained one cycle earlier from the feedback counter. B SAVE COUNT<1:0> is applied to both X and Y reload decoders. The appropriate decoder is enabled by the mask double-bit error signal. The decoder decodes the B SAVE COUNT<1:0> input and asserts a reload signal to the appropriate bit-storage block. The reload signal loads the negated write-enable signal into the location corresponding to the location of the bad data in the data buffer.

2.7.2 Unloading Data From the MDB

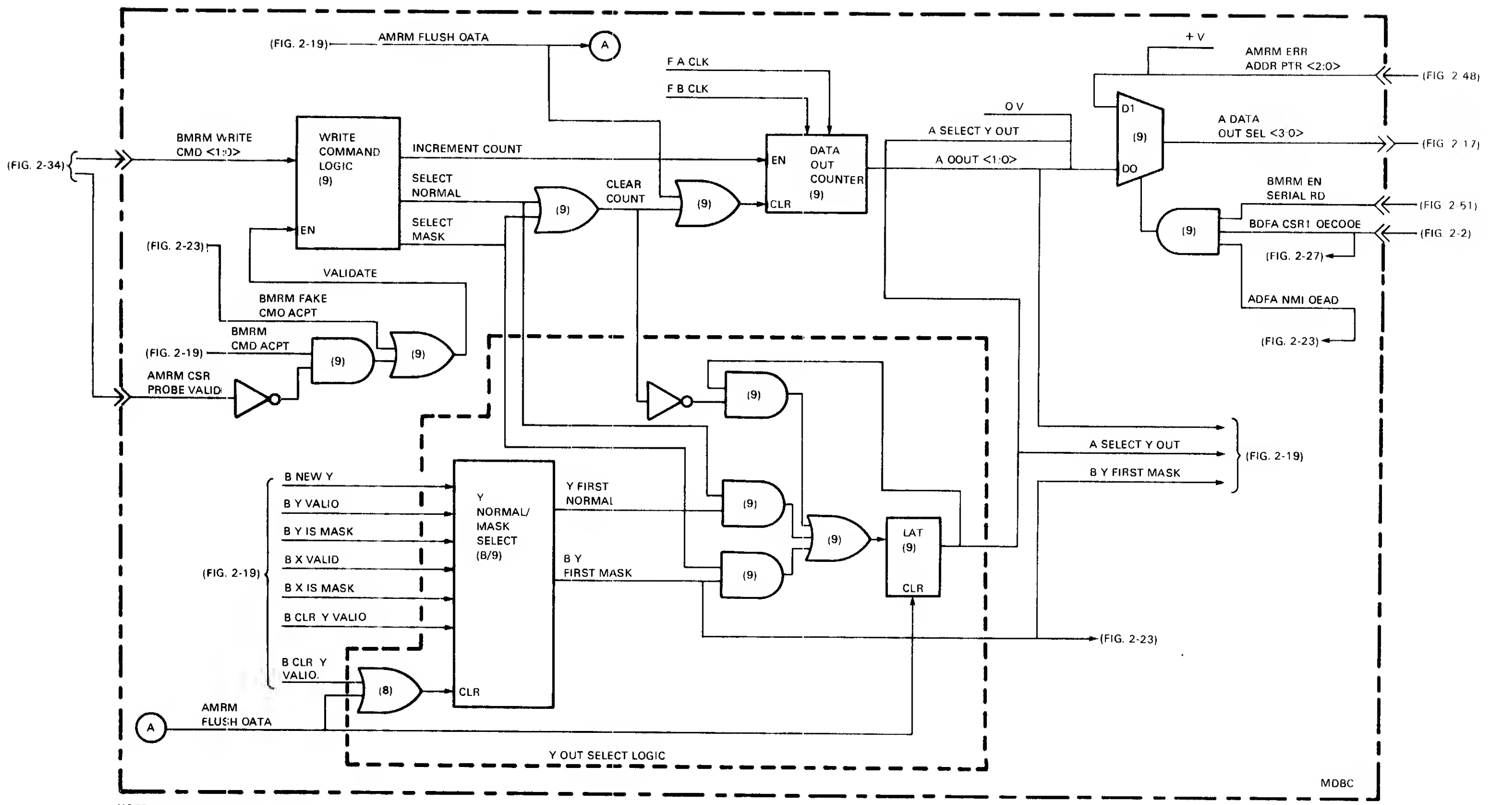
2.7.2.1 General -- Figure 2-25 is a block diagram of the MDBC data out selection function. The major areas shown in Figure 2-25 are defined below.

- Data-Out Counter -- provides a two-bit select output (A DOUT<1:0>) used as the two least significant bits of the A DATA OUT SEL<3:0> output selection signal. Also used to select the output signals from the X and Y bit-storage areas. When enabled, the counter is incremented each bus cycle.

- Y Out Select Logic -- selects the Y buffer for an unload of write data from the data buffer by determining the second most-significant bit of A DATA OUT SEL<3:0>. Also selects the Y buffer for a load of feedback data into the data buffer by determining the most-significant bit of B FB SEL<2:0>.

- Write Command Logic -- decodes input write commands to:

  -- Initiate an unload of normal data
  -- Initiate an unload of masked data
  -- continue an unload that has already been initiated



NOTE:
NUMBER DESIGNATIONS IN PARENTHESES
REFER TO ENGINEERING DRAWINGS
CONTAINING CORRESPONDING LOGIC.

Figure 2-25 MDBC -- MDB Data-Out Selection

2.7.2.2 Detailed -- A DATA OUT SEL<3:0> is the select signal that selects the MDB output that passes through the data-read port. A DATA OUT SEL<3:0> is obtained from a mux which can choose a select signal from the MRM (AMRM ERR ADDR PTR<2:0>) via its D1 input, or the MDBC select signal via its D0 input.

The AMRM ERR ADDR PTR<2:0> input is selected when a read of CSR1 is executing (BMRM EN SERIAL RD and BDFA CSR1 DECODE both true) and the NMI is quiet (ADFA NMI DEAD true). The most-significant bit of A DATA OUT SEL<3:0> is a 1, hence the MDB address buffer is accessed. The three least-significant bits select the location of the CSR1 bits in the address buffer.

For normal operation, the mux chooses the MDBC select bits. The most-significant bit of A DATA OUT SEL<3:0> is a 0, hence the MDB data buffer is accessed. A SELECT Y OUT is the next most-significant bit and determines whether the X or Y buffer is selected. The last two bits of A DATA OUT SEL<3:0> are A DOUT<1:0> obtained from the data out counter. The counter receives a CLEAR COUNT and an INCREMENT COUNT signal from the write-command logic. CLEAR COUNT resets the counter to zero and INCREMENT COUNT enables the counter to be incremented each bus cycle by F A CLK and F B CLK.

The write-command logic receives a two-bit write command (BMRM WRITE CMD<1:0>) from the MRM which specifies the function being executed as shown in Table 2-7.

Table 2-7 Write Commands

BMRM WRITE CMD <1	0>	Output*	Function
0	0	-----	No Op
0	1	INCREMENT COUNT	Next longword
1	0	SELECT MASK	Select first masked longword
1	1	SELECT NORMAL	Select first normal longword

* If VALIDATE is true.

The write command is decoded by the write-command logic and asserts an output according to Table 2-7, if VALIDATE is true. VALIDATE is true when the MRM asserts BMRM CMD ACPT while AMRM CSR PROBE VALID is false, or when the MRM asserts BMRM FAKE CMD ACPT for maintenance testing.

The write command received for a normal (non-masked) octaword write is SELECT NORMAL followed by three INCREMENT COUNTs. SELECT NORMAL initiates an unload of normal data from the data buffer. SELECT NORMAL is applied to the Y-out-select logic where it checks the state of Y FIRST NORMAL to determine if the Y buffer should be accessed. If Y FIRST NORMAL is true, A SELECT Y OUT asserts, making A DATA OUT SEL<2> a 1 to select the Y buffer for the unload. If Y FIRST NORMAL is false, the X buffer is selected for the unload. The operation of the Y-out-select logic is discussed in Section 2.7.3.

SELECT NORMAL asserts CLEAR COUNT which resets the data-out counter to zero. The three INCREMENT COUNT commands that follow, enable the data-out counter to be incremented. This increments A DOUT<1:0> and then A DATA OUT SEL<1:0> to step through the four locations of the selected buffer.

The write command received for a masked octaword write is SELECT MASK followed by three INCREMENT COUNTs. SELECT MASK initiates an unload of masked data from the data buffer. SELECT MASK is applied to the Y-out-select logic where it checks the state of B Y FIRST MASK to see if the Y buffer should be accessed. If B Y FIRST MASK is true, A SELECT Y OUT asserts, making A DATA OUT SEL<2> a 1 to select the Y buffer for the unload. If B Y FIRST MASK is false, the X buffer is selected for the unload.

SELECT MASK asserts CLEAR COUNT which resets the data-out counter to zero. The three INCREMENT COUNT commands that follow, enable the data-out counter to be incremented. This increments A DOUT<1:0> and then A DATA OUT SEL<1:0> to step through the four locations of the selected buffer.

A DOUT<1:0> and A SELECT Y OUT are used to unload the good-data and write-enable bits from the X and Y bit-storage block (see Figure 2-19). A DOUT<1:0> is used to select the good-data and write-enable bits from their locations within the X and Y storage blocks (see Figure 2-22). A SELECT Y OUT is used to choose the X bits or the Y bits. If A SELECT Y OUT is true, B Y STRD WE and the inverse of B Y STRD GOOD DATA are selected and become BMDP WRITE ENABLE and BMDP BAD DATA respectively. BMDP WRITE ENABLE and BMDP BAD DATA output the MBDC and are translated from ECL to TTL to become AMCL WRITE ENABLE and AMCL BAD DATA respectively, for the NAB. If A SELECT Y OUT is false, the X write-enable and good-data bits are selected.

The data parity bits from the MDB (B BYTE PARITY<3:0>) are combined with the write-enable and bad-data bits in a parity generator which generates a composite parity bit (BMDP DATA PARITY). BMDP DATA PARITY is translated from ECL to TTL to become AMCL DATA PARITY for the NAB. BMRM FORCE BAD DPAR from the MRM is used to generate a parity error for maintenance testing.

2.7.3 Y Out Select Logic

2.7.3.1 General -- The Y-out-select logic selects the buffer (X or Y) that is to be unloaded by controlling the A DATA OUT SEL<2> bit of the select signal. When the logic asserts A SELECT Y OUT, the A DATA OUT SEL<2> bit becomes a 1 and the Y buffer is selected. If the logic does not assert its A SELECT Y OUT output, the A DATA OUT SEL<2> bit becomes a 0 and the X buffer is selected.

As previously mentioned, the X buffer is used whenever possible. The Y buffer is used only if the X buffer already has data.

If the X and Y buffers contain the same type of data, the Y-out-select logic acts as a FIFO and the data loaded in first will be unloaded first. If the buffers have a mix of data (normal data in one buffer and masked data in the other), the MRM can unload either buffer by commanding a mask write or a normal write to the Y-out-select logic via the write-command logic.

Figure 2-26 is a flow diagram of the Y-out-select logic. The flow starts with the condition that there is data in the X buffer and new data arrives to be placed into the Y buffer. It then determineds what type of data is in each buffer. After that, the four possible conditions

- Normal data in both buffers
- Normal data in X, masked data in Y
- Masked data in X, normal data in Y
- Masked data in both buffers)

are flowed to empty both buffers. In the following discussion of the flow diagram, note that all of the signals required by the flow are input to the Y-out-select logic in Figure 2-25.

2.7.3.2 **Detailed** -- With data already in the X buffer (B X VALID true), new data is placed into the Y buffer (B NEW Y asserts). The logic then checks B Y IS MASK and B X IS MASK to determine the type of data in each buffer.

If the new Y data is masked but the X data is not, the Y buffer select block asserts B Y FIRST MASK indicating that the Y buffer will be selected by a masked write command. If a masked write occurs (SELECT MASK asserts), A SELECT Y OUT is asserted to select the Y buffer and the Y bit-storage block output. When the Y buffer is emptied, B CLR Y VALID is asserted by the full logic and resets the Y-out-select logic (negating B Y FIRST MASK). The X buffer is then unloaded by a SELECT NORMAL command. If a normal write occured (instead of a masked write), SELECT NORMAL asserts and the X buffer is unloaded. When SELECT MASK does assert, A SELECT Y OUT asserts and selects the Y buffer for the unload. B CLR Y VALID then resets the Y-out-select logic.

For the case where there is masked data in both buffers, the Y buffer select block does not assert either of its outputs thereby selecting the X buffer for the next unload. SELECT MASK unloads the X buffer after which B CLR X VALID is asserted by the full logic. The Y buffer select block senses the assertion of B CLR X VALID as an indication that the X buffer has been unloaded. It also senses the true state of B Y VALID as an indication that

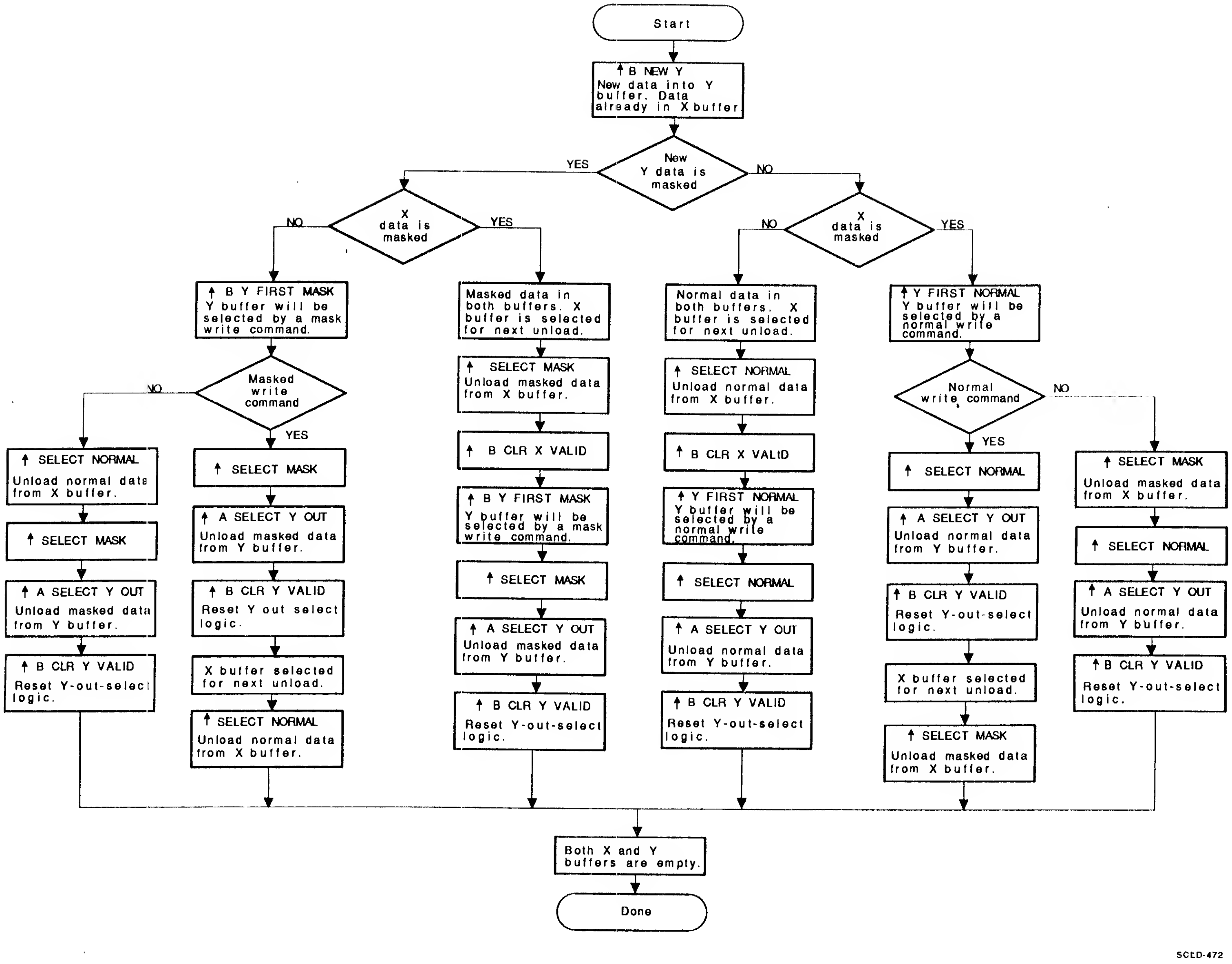

Start
↑ B NEW Y
New data into Y buffer. Data already in X buffer
New Y data is masked
YES
NO
X data is masked
NO
YES
↑ B Y FIRST MASK
Y buffer will be selected by a mask write command.
Masked write command
NO
YES
↑ SELECT NORMAL
Unload normal data from X buffer.
↑ SELECT MASK
↑ A SELECT Y OUT
Unload masked data from Y buffer.
↑ B CLR Y VALID
Reset Y-out-select logic.
↑ SELECT MASK
↑ A SELECT Y OUT
Unload masked data from Y buffer.
↑ B CLR Y VALID
Reset Y out select logic.
X buffer selected for next unload.
↑ SELECT NORMAL
Unload normal data from X buffer.
Masked data in both buffers. X buffer is selected for next unload.
↑ SELECT MASK
Unload masked data from X buffer.
↑ B CLR X VALID
↑ B Y FIRST MASK
Y buffer will be selected by a mask write command.
↑ SELECT MASK
↑ A SELECT Y OUT
Unload masked data from Y buffer.
↑ B CLR Y VALID
Reset Y-out-select logic.
X data is masked
NO
YES
Normal data in both buffers. X buffer is selected for next unload.
↑ SELECT NORMAL
Unload normal data from X buffer.
↑ B CLR X VALID
↑ Y FIRST NORMAL
Y buffer will be selected by a normal write command.
↑ SELECT NORMAL
↑ A SELECT Y OUT
Unload normal data from Y buffer.
↑ B CLR Y VALID
Reset Y-out-select logic.
↑ Y FIRST NORMAL
Y buffer will be selected by a normal write command.
Normal write command
YES
NO
↑ SELECT NORMAL
↑ A SELECT Y OUT
Unload normal data from Y buffer.
↑ B CLR Y VALID
Reset Y-out-select logic.
X buffer selected for next unload.
↑ SELECT MASK
Unload masked data from X buffer.
↑ SELECT MASK
Unload masked data from X buffer.
↑ SELECT NORMAL
↑ A SELECT Y OUT
Unload normal data from Y buffer.
↑ B CLR Y VALID
Reset Y-out-select logic.
Both X and Y buffers are empty.
Done
SCLD-472

Figure 2-26 Y Out Select Flow Diagram

there is data in the Y buffer. The Y buffer select block then asserts B Y FIRST MASK so that the next masked write command will select the Y buffer. When SELECT MASK asserts, A SELECT Y OUT asserts to select the Y buffer for the unload. After the Y buffer is unloaded, B CLR Y VALID asserts to reset the Y-out-select logic.

For the case where the new Y data is not masked but the X data is, the Y buffer select block asserts Y FIRST NORMAL indicating that the Y buffer will be selected by a normal write command. If a normal write occurs (SELECT NORMAL asserts), A SELECT Y OUT is asserted to select the Y buffer and the Y bit-storage block output. When the Y buffer is emptied, B CLR Y VALID is asserted by the full logic and resets the Y-out-select logic (negating Y FIRST NORMAL). The X buffer is then unloaded by a SELECT MASK command. If a masked write occured (instead of a normal write), SELECT MASK asserts and the X buffer is unloaded. When SELECT NORMAL does assert, A SELECT Y OUT asserts and selects the Y buffer for the unload. After the Y buffer is unloaded, B CLR Y VALID is asserted and resets the Y-out-select logic.

For the case where there is normal data in both buffers, the Y buffer select block does not assert either of its outputs thereby selecting the X buffer for the next unload. SELECT NORMAL unloads the X buffer after which B CLR X VALID is asserted by the full logic. The Y buffer select block senses the assertion of B CLR X VALID as an indication that the X buffer has been unloaded. It also senses the true state of B Y VALID as an indication that there is data in the Y buffer. The Y buffer select block then asserts Y FIRST NORMAL so that the next normal write command will select the Y buffer. When SELECT NORMAL asserts, A SELECT Y OUT asserts to select the Y buffer for the unload. After the Y buffer is unloaded, B CLR Y VALID is asserted and resets the Y-out-select logic.

B Y FIRST MASK from the Y buffer select block becomes B FB SEL<2> (Figure 2-23) which is the most-significant bit of the feedback select signal. B FB SEL<2> selects which buffer will receive the feedback data. Note in the flow diagram that B Y FIRST MASK asserts in only two cases:

- When the X buffer has normal data and the Y buffer has just received masked data. In this case the feedback data should go into the Y buffer to overwrite the masked data.

- When there is masked data in both buffers and the X buffer has just been emptied. When there is masked data in both buffers, the MRM will always unload the X buffer before it reads feedback data to overwrite the masked data in the Y buffer.

2.7.4 Internal Error, Write Decode RAM, and Clocks

Miscellaneous functions executed on the MDBC MCA are shown in Figure 2-27.

The internal error logic generates a delayed error signal for the MRM. AMRM INT ERROR from the MRM enables a counter to be incremented each bus cycle by F A CLK and F B CLK. After 15 1/2* bus cycles, the counter outputs TERMINAL COUNT causing BMDP DLY INT ERR to assert back to the MRM. BMDP DLY INT ERR is locked up by feedback so long as AMRM INT ERROR remains asserted.

The write-decode-RAM logic generates the write strobe for the decode RAM. If feedback bit <20> is a 1 when the MRM asserts AMRM CSR WRITE and the DAD asserts BDFA CSR1 DECODE, then BMDP DECRAM WRITE asserts as the write strobe for the decode RAM (Figure 2-2).

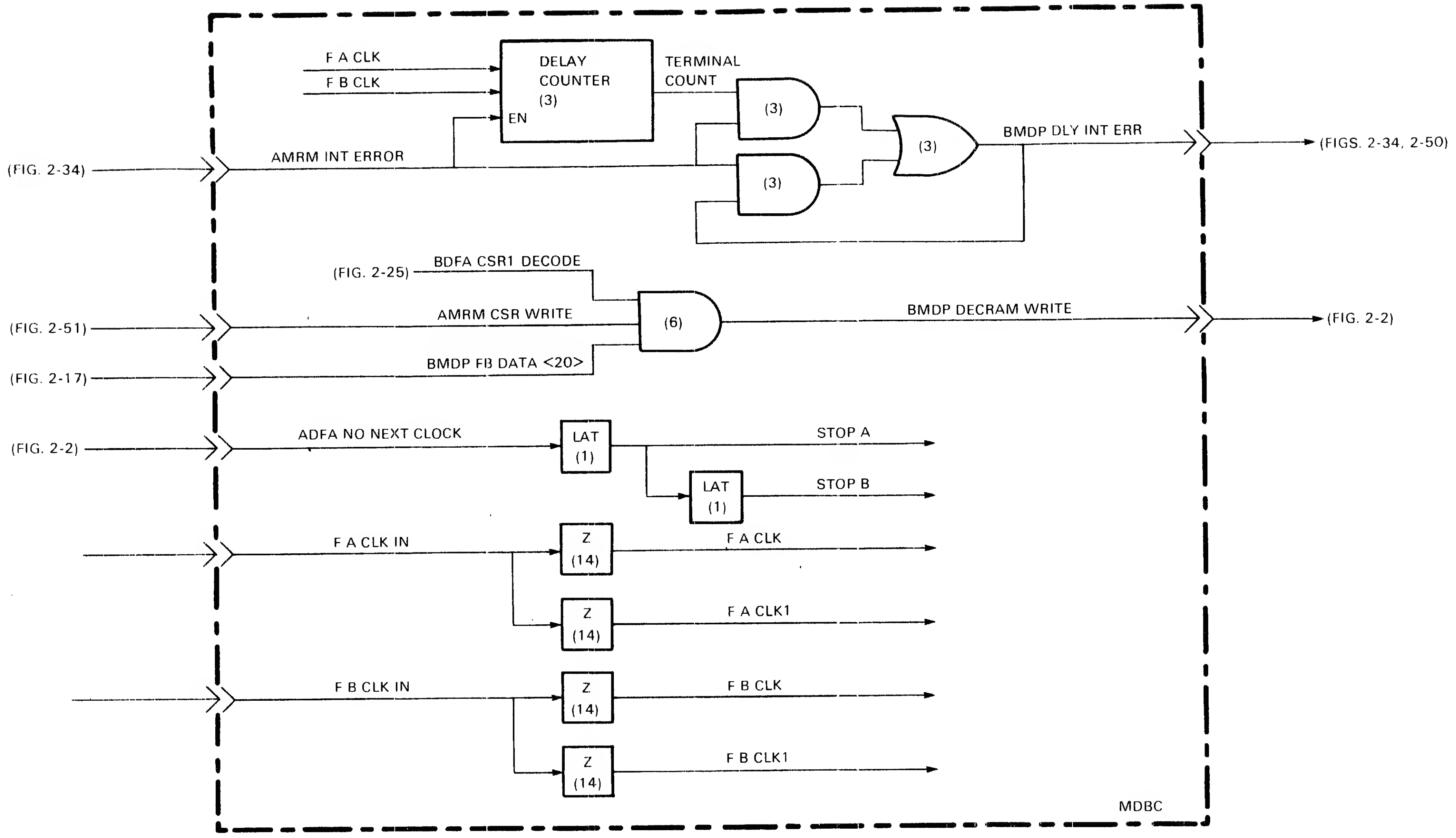
F A CLK IN and F B CLK IN distribute clocks throughout the MDBC.

2.8 DATA CHECK (DCHK) MCA

The DCHK MCA performs the following functions:

- Checks the data returned from the MAR4 for single-bit or double-bit errors.
- Generates correction signals for single-bit errors.
- Reports both single-bit and double-bit errors.
- Detects and reports an asserted bad-data bit in the MDBC by means of the INT BAD DATA bit from the MAR4.
- Detects and reports a data parity error in the MAR4 by means of the INT BAD DATA bit from the MAR4.
- Assembles and formats CSR2 data.
- Provides for reading and writing of CSR2.
- Provides for diagnostic testing.

* Fifteen and one-half cycles is a minimum. The time period could be longer depending on the assertion time of AMRM INT ERROR with respect to F A CLK and F B CLK.



NOTE:
NUMBER DESIGNATIONS IN PARENTHESES
REFER TO ENGINEERING DRAWINGS
CONTAINING CORRESPONDING LOGIC.

MKV86-1293

Figure 2-27 MDBC -- Internal Error, Write Decode RAM, and Clocks

Figure 2-28 is a block diagram of the DCHK. The block diagram illustrates four functional areas that have been defined as distinct logic areas by the engineering documentation (logic prints, etc.). This description follows the same functional partioning. The four functional areas are defined below.

- Syndrome Generate -- generates error syndromes by generating check bits from the read data and comparing them to the check bits returned from the MAR4.
- Error Check -- using the generated syndrome bits, detects a single-bit error, a double-bit error, or an asserted INT BAD DATA bit from the MAR4. An asserted INT BAD DATA from the MAR4 means that the bad-data bit was set in the MDBC, or a data parity error was detected in the MAR4.
- Error Status -- locks up any errors detected by the error-check logic as CSR2 error bits. Generates and locks up a high-error-rate signal as a CSR2 error bit. Generates a LD PAGE ADDR signal which asserts an NMI interrupt via the ARID.
- Serializer -- does parallel-to-serial conversion of CSR2 data for a CSR2 read operation.

2.8.1 Syndrome Generate

The ECC check bits associated with the read data are applied to the DCHK as ARCV CHECK BITS<6:0>. The check bits are passed through a mux and then loaded into a latch by BMRM NEW LW. The latched check bits are applied to the syndrome generate as CHECK BITS<6:0>.

BMRM NEW LW also loads a copy of the check bits into another latch where they output as MEM CB<6:0>. MEM CB<6:0> is sent to the serializer where it becomes part of CSR2 (except in diagnostic mode; see Section 2.8.5).

Data read (fed back) from the MAR4 is input to the DCHK as BMDP FB DATA<31:0>. BMDP FB DATA<31:0> becomes B DATA IN<31:0> and then B DATA<31:0> which is applied to the syndrome generate logic.

In the syndrome generate logic, a check-bit generator uses the data bits to regenerate the seven ECC check bits. The regenerated check bits are compared to CHECK BITS<6:0> in a comparator which outputs seven syndrome bits (SYNDROME<6:0>) to the error check logic. If the data longword is error-free, SYNDROME<6:0> are all 0s.

2.8.2 Error Check

The syndrome bits (SYNDROME<6:0>) are applied to the error check logic (Figure 2-29) where they are input to an INT-bad-data detector. The detector looks for the syndrome code that specifies

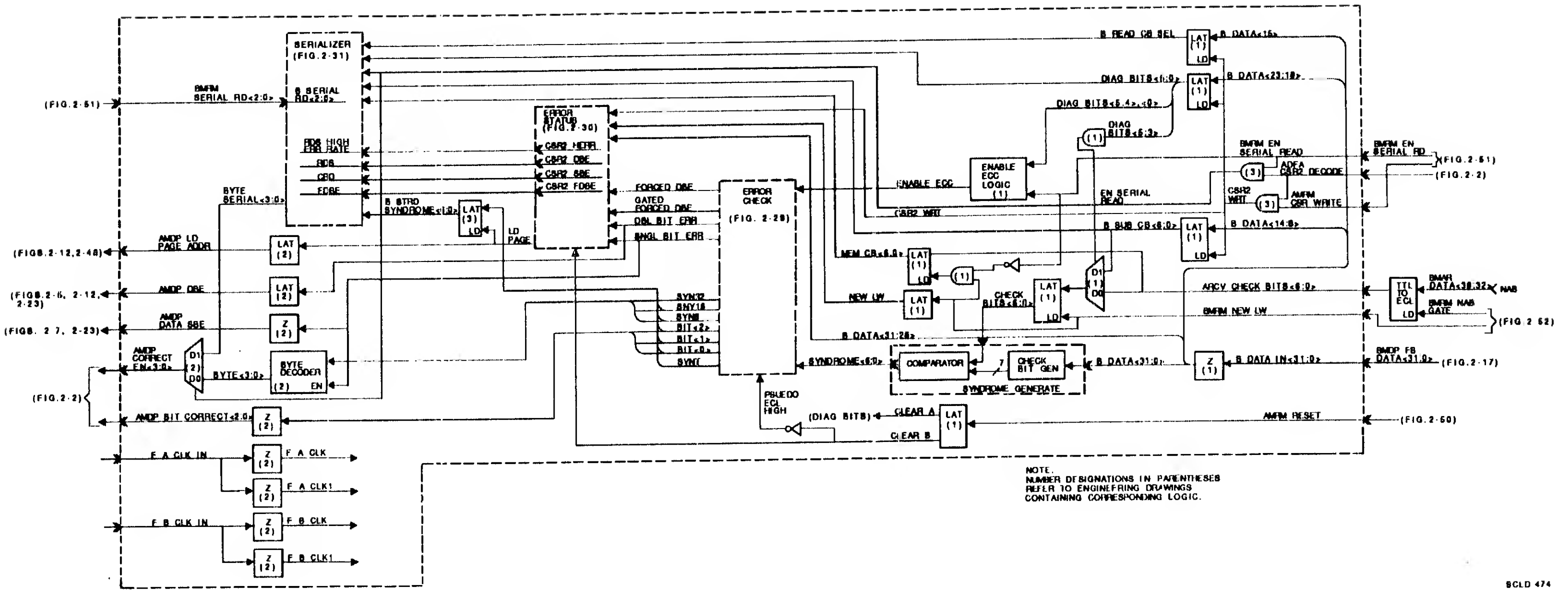


Figure 2-28 DCHK Block Diagram

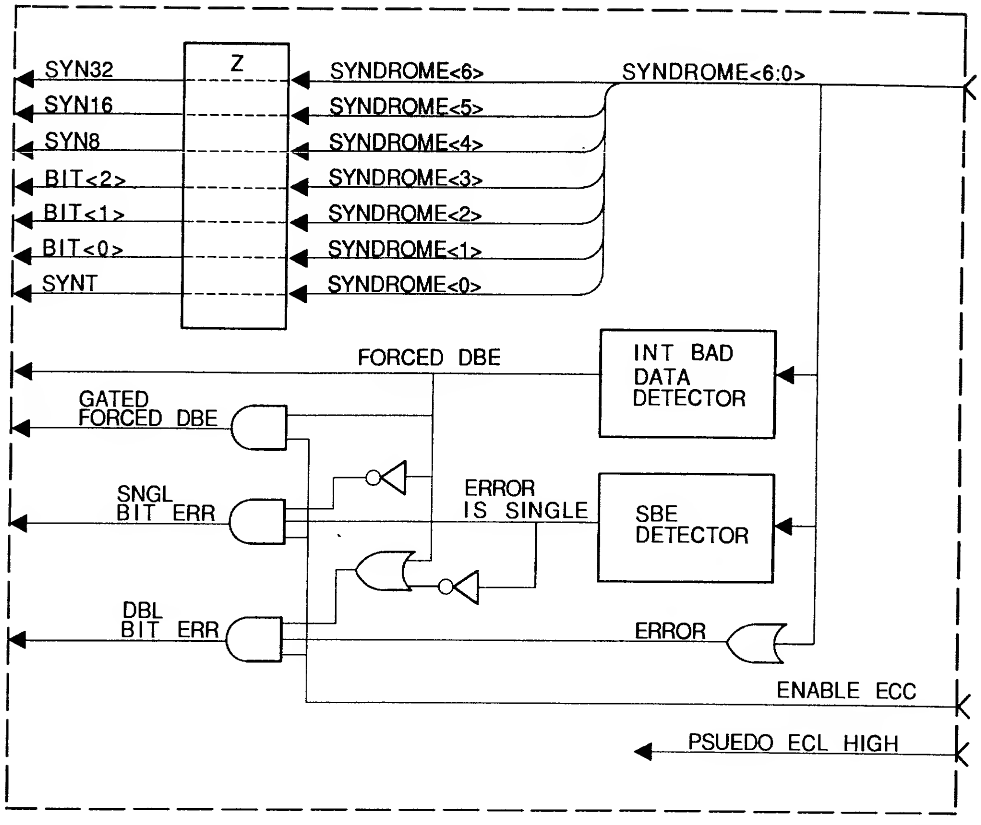


Figure 2-29 Error Check Block Diagram

that the INT BAD DATA bit* in the MAR4 was set. When the detector senses the INT-bad-data syndrome code (0010011), it asserts FORCED DBE (double-bit error) as an output of the error-check logic.

FORCED DBE causes GATED FORCED DBE to assert as another error-check output when ENABLE ECC is true. During normal operation, ENABLE ECC is asserted by the enable-ECC logic which is under diagnostic control (Section 2.8.5).

SYNDROME<6:0> are applied to an SBE (single-bit error) detector which looks for those syndrome codes that indicate a single-bit error has occurred+. When the detector senses a single-bit error

* The MAR4 INT BAD DATA bit is set by an asserted bad-data bit in the MCL (MDBC) or by the detection of a data parity error in the MAR4.

+ A single-bit error is indicated by an odd number of asserted syndromes except for syndrome code 0000111. This code is obtained when all the data bits and check bits are 1s, which is a double-bit error condition. The SBE detector does not respond to this code, hence a double-bit error is indicated.

code, it asserts ERROR IS SINGLE which in turn asserts SNGL BIT ERR if ENABLE ECC is true. If the INT BAD DATA bit is set, FORCED DBE will inhibit the assertion of SNGL BIT ERR. Hence, the assertion of SNGL BIT ERR indicates a single-bit error with the INT BAD DATA bit reset.

SNGL BIT ERR outputs the DCHK as AMDP DATA SBE (Figure 2-28). AMDP DATA SBE is sent to the ARID where it is used for parity correction.

A double-bit error is detected in the error-check logic by first ORing the seven syndrome bits. The OR gate asserts ERROR if any error condtion exists (syndrome bits are all 0s for a no-error condition). The assertion of ERROR will assert DBL BIT ERR if ENABLE ECC is true and:

- The INT BAD DATA bit is set (FORCED DBE true), or
- The error is not a single-bit error (ERROR IS SINGLE false).

Hence, the assertion of DBL BIT ERR means a double-bit error in the current longword or, the INT BAD DATA bit is set.

DBL BIT ERR outputs the DCHK as AMDP DBE which is sent to the ARID and FUNK MCAs. In the ARID, AMDP DBE is used to generate a memory interrupt on the NMI. In the FUNK, it is used to signify bad data in the read/return or read/continue function codes.

When a single-bit error exists, SYNDROME<6:4> points to the byte containing the error. SYNDROME<6:4> outputs the error-check logic as SYN32, SYN16, and SYN8 respectively. They are applied to a byte decoder which, when enabled, decodes the bits to assert BYTE<X> where X is the byte with the erroneous bit. BYTE<X> outputs the DCHK (via a mux) as AMDP CORRECT EN<X> which enables the correction logic within the appropriate DAD MCA. The byte decoder is enabled by SNGL BIT ERR when a single-bit error has been detected.

For the single-bit error case, SYNDROME<3:1> indicates, in binary format, which data bit within the erroneous byte is incorrect. SYNDROME<3:1> outputs the error-check logic as BIT<2:0> respectively. BIT<2:0> outputs the DCHK as AMDP BIT CORRECT<2:0> and are applied to the DAD error correction logic.

2.8.3 Error Status

Error signals GATED FORCED DBE, SNGL BIT ERR, and DBL BIT ERR are coupled from the error-check logic to the error-status logic (Figure 2-30) where they are locked up as status bits for CSR2.

The three error signals are enabled into the error-status logic by NEW LW (derived from BMRM NEW LW).

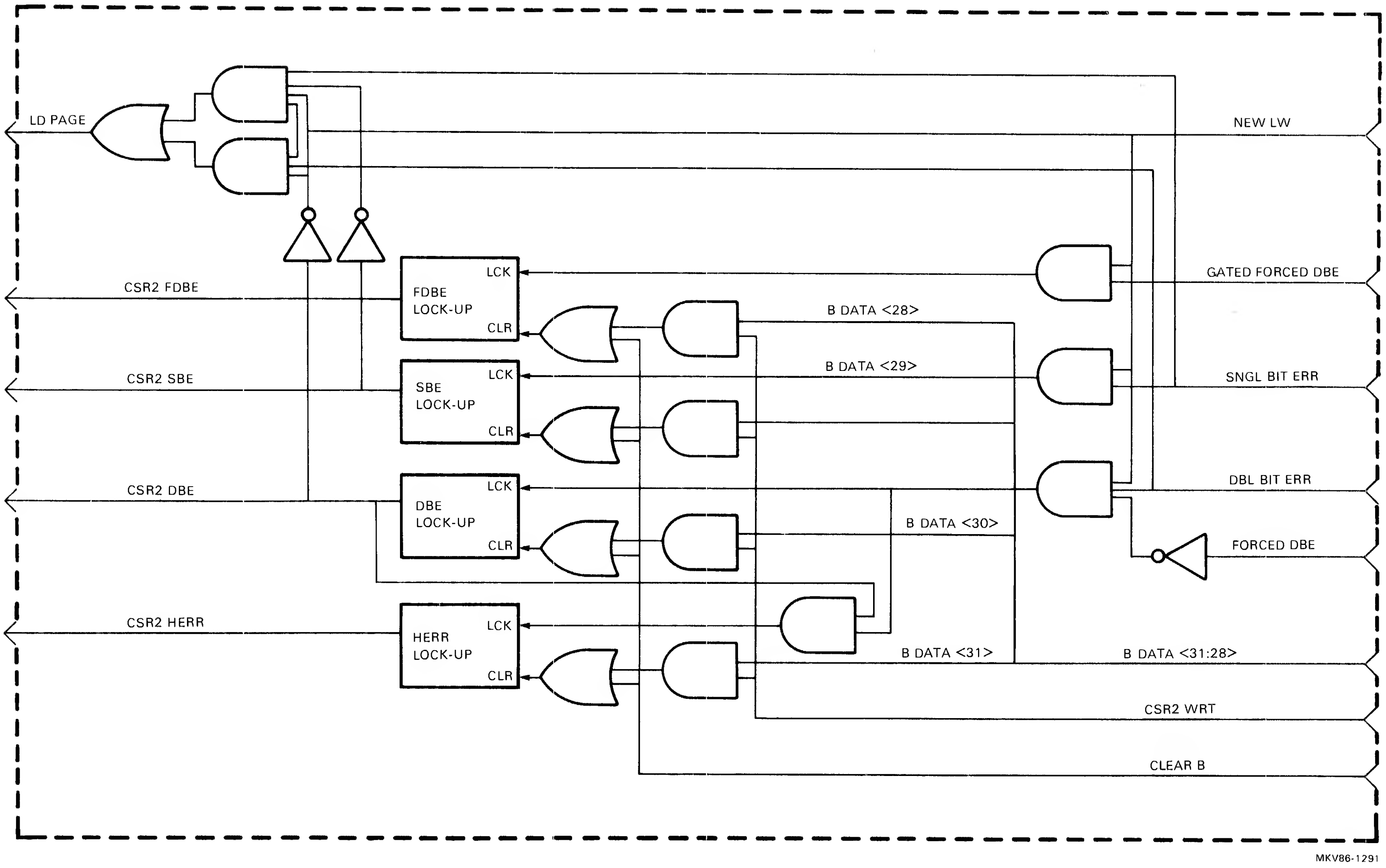


Figure 2-30 Error Status Block Diagram

GATED FORCED DBE is applied to the FDBE lock-up block which outputs CSR2 FDBE.

SNGL BIT ERR is applied to the SBE lock-up block which outputs CSR2 SBE.

DBL BIT ERR is applied to the DBE lock-up block (if FORCED DBE is false), which outputs CSR2 DBE. CSR2 DBE is asserted only if FORCED DBE is false, hence the assertion of CSR2 DBE indicates a true double-bit error, not an asserted INT BAD DATA bit.

CSR2 DBE is applied to an AND gate which looks for another double-bit error input from the error-check logic. If another double-bit error occurs while CSR2 DBE is still locked up, an input is asserted to the HERR (high error rate) lock-up block which outputs CSR2 HERR.

The four lock-up blocks are cleared by a write to CSR2 with the appropriate data bit (B DATA<31:28>) asserted. CSR2 WRT is asserted by the assertion of AMRM CSR WRITE from the MRM while ADFA CSR2 DECODE is asserted by the DAD.

The four CSR2 error outputs are applied to the serializer as CSR2 error bits.

A LD PAGE error signal is generated by the error-status logic when a single- or double-bit error occurs. LD PAGE is asserted for only one bus cycle. When the single- or double-bit error is locked up (CSR2 SBE or CSR2 DBE asserts), LD PAGE is negated.

LD PAGE is sent to the MRM as AMDP LD PAGE ADDR where it holds the error page address pointer. This reserves the location in the MDB that contains the address at which the single-bit or double-bit error occurred.

AMDP LD PAGE ADDR is also sent to the ARID where it generates an interrupt on the NMI. Single-bit errors are corrected on the fly and do not halt the current operation. They do generate interrupts and are logged by the software so that if a location gets too many single-bit errors, the software can remove the page from its list of available pages.

Double-bit errors are hard errors that halt the current operation. Hence, as seen in Figure 2-30, LD PAGE is not asserted by a single-bit error if a double-bit error has already been detected.

In addition, LD PAGE saves the syndrome bits from the error-check logic by loading them into a latch. The latched syndrome bits (B STRD SYNDROME<6:0>) are applied to the serializer as CSR2 bits.

2.8.4 Serializer and CSR2

The serializer logic (Figure 2-31) assembles and formats the CSR2 data. It functions as a parallel-to-serial converter when a read of CSR2 occurs. Figure 2-32 is a bit map of CSR2.

Four parallel-to-serial converters each supply a byte of CSR2 data. The converters are enabled by a binary incrementer which is itself enabled by EN SERIAL READ. EN SERIAL READ is asserted by the assertion of BMRM EN SERIAL READ from the MRM when ADFA CSR2 DECODE from the DAD is true.

When enabled, the incrementer responds to the three-bit input code (BMRM SERIAL RD<2:0>) received from the MRM. The three-bit input code is incremented in binary format causing the output of the incrementer (B CNT<2:0>) to also increment in binary format. B CNT<2:0> enables the parallel inputs to the converters to pass to the output in serial format. Eight cycles are required to step through the CNT<2:0> bits and read out a byte from each converter.

The first converter outputs BYTE SERIAL<3>. This converter receives the four error signals locked up by the error-status logic. Note the change in signal names going from the error-check logic to the error-status logic. As byte three of CSR2 contains only four information bits, only two counter bits (B CNT<1:0>) are supplied by the incrementer.

The second converter receives six diagnostic bits (DIAG<5:0>) which are output as BYTE SERIAL<2>.

The third converter receives seven check bits (CSR2 RD CB<6:0>) and B READ CB SEL which are output as BYTE SERIAL<1>. The check bits are obtained from a mux which selects a copy of the check bits received from the MAR4 (MEN CB<6:0>), or the substitute check bits (B SUB CB<6:0>) used for diagnostics. The copy of the MAR4 check bits is latched by the assertion of BMRM EN SERIAL READ when CSR2 is being read.

Mux selection is made by B READ CB SEL which asserts in diagnostic mode (Section 2.8.5). B READ CB SEL is also one of the byte <1> bits of CSR2.

The fourth converter receives the seven stored syndrome bits (B STRD SYNDROME<6:0>) which are output as BYTE SERIAL<0>.

The CSR2 serial data (BYTE SERIAL<3:0>) is mux selected for the AMDP CORRECT EN<3:0> output when a read of CSR2 is occurring (EN SERIAL READ true).

2.8.5 Diagnostic Mode (Figure 2-28)

Most of the MCL diagnostic bits reside in the DCHK. To run the diagnostics, a write of CSR2 is done wherein control bits and substitute check bits are loaded into three DCHK latches. The data

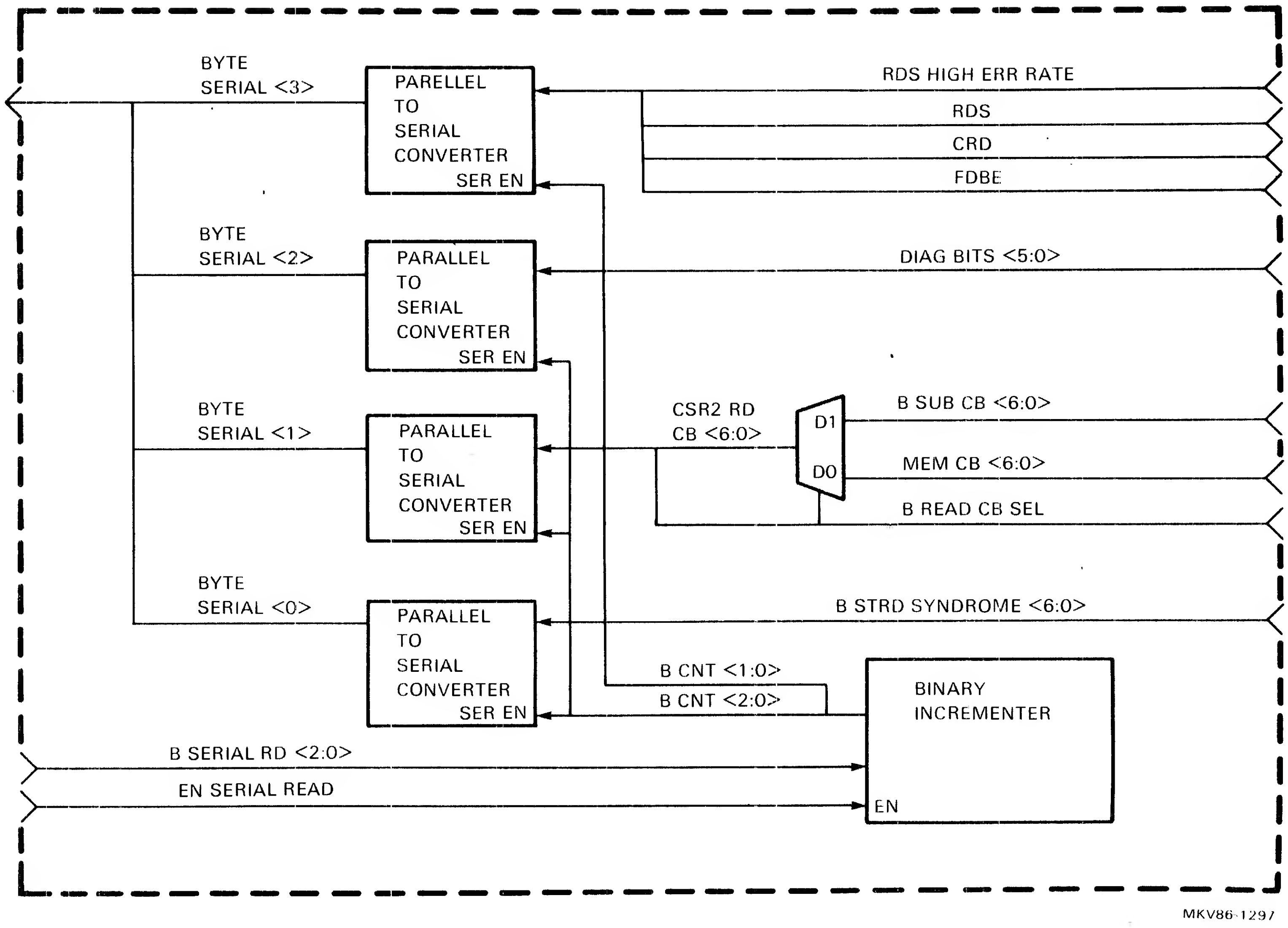


Figure 2-31 Serializer Block Diagram

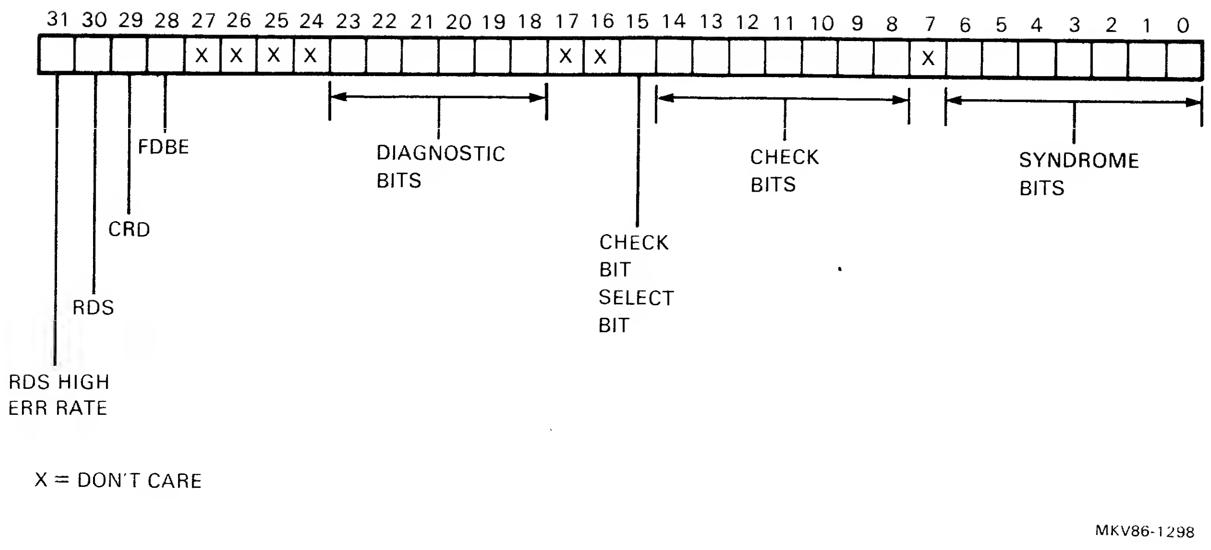


Figure 2-32 CSR2 Bit Map

lines are used to carry the diagnostic bits into the latches. When CSR2 WRT asserts:

1. Diagnostic check bits B DATA<14:8> are loaded into a latch which outputs B SUB CB<6:0>. B SUB CB<6:0> are applied to a mux in the serializer where they are selected as part of byte <1> of CSR2. B SUB CB<6:0> are also applied to another mux where they are selected as the substitute (test) check bits for the syndrome-generate logic.

2. Select bit B DATA<15> is loaded into a latch which outputs B READ CB SEL. B READ CB SEL is sent to the serializer mux where it selects the diagnostic check bits (B SUB CB<6:0>) for byte <1> of CSR2. The select bit is included as part of byte <1>.

3. Diagnostic control bits B DATA<23:18> are loaded into a latch which outputs DIAG BITS<5:0>. DIAG BITS<5:0> are applied to the serializer as byte <2> of CSR2. In addition, DIAG BITS<5:0> performs the following:

   - DIAG BITS<5> -- diagnostic mode enable bit. Places the MCL into the diagnostic mode. Within the DCHK, it is applied to the enable-ECC logic and to an AND gate which looks at diagnostic bit <3>.

   - DIAG BITS<4> -- in diagnostic mode (DIAG BITS<5> true) with loopback disabled (DIAG BITS<0> false), disables the error-check logic by negating ENABLE ECC. Disabling the error-check logic inhibits the correction of read data.

- DIAG BITS<3> -- in diagnostic mode, selects substitute check bits (B SUB CB<6:0>) for the syndrome generate logic, instead of check bits from the MAR4.

- DIAG BITS<2:1> -- forces the MRM to generate a parity error on the NAB via the MDBC. The MRM receives the diagnostic bits via CSR2.

- DIAG BITS<0> -- in diagnostic mode, enables a loopback of data from the MDB into the MCL read-data path for testing. In the DCHK, it enables the error-check logic by asserting ENABLE ECC.

The enable-ECC logic asserts ENABLE ECC to the error-check logic in normal mode except when CSR2 is being read. When CSR2 is being read, BMRM EN SERIAL READ asserts and negates the ENABLE ECC output. This is done so the outputs of the error-check logic will not change (and thereby change the CSR2 bits) during the eight cycles required to serially read CSR2.

In diagnostic mode, the enable-ECC logic is controlled by the diagnostic bits as shown in Truth Table 2-8. The four states shown in the table are described below.

1. Normal mode -- ENABLE ECC is a function of BMRM EN SERIAL READ.

2. Diagnostic mode with loopback enabled -- ECC is enabled.

3. Diagnostic mode with loopback disabled -- DIAG BITS<4> negated -- ENABLE ECC is a function of BMRM EN SERIAL READ.

4. Diagnostic mode with loopback disabled -- DIAG BITS<4> asserted -- disables ECC.

2.8.6 Reset and Clocks (Figure 2-28)

CLEAR A and CLEAR B are asserted in the DCHK by AMRM RESET from the MRM. CLEAR A clears the three latches containing the diagnostic control bits, the diagnostic check bits, and the CSR2 check-bit select signal.

CLEAR B clears the four CSR2 error bits in the error-status logic. It also negates PSUEDO ECL HIGH in the error-check logic.

F A CLK IN and F B CLK IN supply A and B clocks to the DCHK logic.

Table 2-8 ENABLE ECC Truth Table

State Number	DIAG BITS<5>	DIAG BITS<0>	DIAG BITS<4>	BMRM EN SERIAL READ	ENABLE ECC
1	0 (Normal Mode)	X	X	0	1
2	1 (Diagnostic Mode)	1 (Enable Loopback)	X	X	1
3	1 (Diagnostic Mode)	0 (Disable Loopback)	0	0	1
4	1 (Diagnostic Mode)	0 (Disable Loopback)	1	X	0

X = don't care

2.9 MRM OVERVIEW (Figure 2-33)

The MRM is comprised of the following:

- MSC = memory sequence control -- an MCA that receives and buffers command information from the DFA. It also checks for internal errors, generates the command field for the memory arrays, and generates the write commands for the MDP.

- MSC1 = memory sequence control 1 -- this MCA is an extension of the MSC MCA. It stores the mask fields associated with two masked write commands, and generates the load enabling commands for the two write ports of the MDB. It also generates the select bits that place the command address into and select the address out of the MDB.

- MASC = memory array status control -- an MCA that checks status of the array boards and selects the board to receive the current command.

- RCS = read control sequencer -- an MCA that controls the transfer of read data from the array boards after the arrays have been read. It generates read commands that specify the type of read data being transferred.

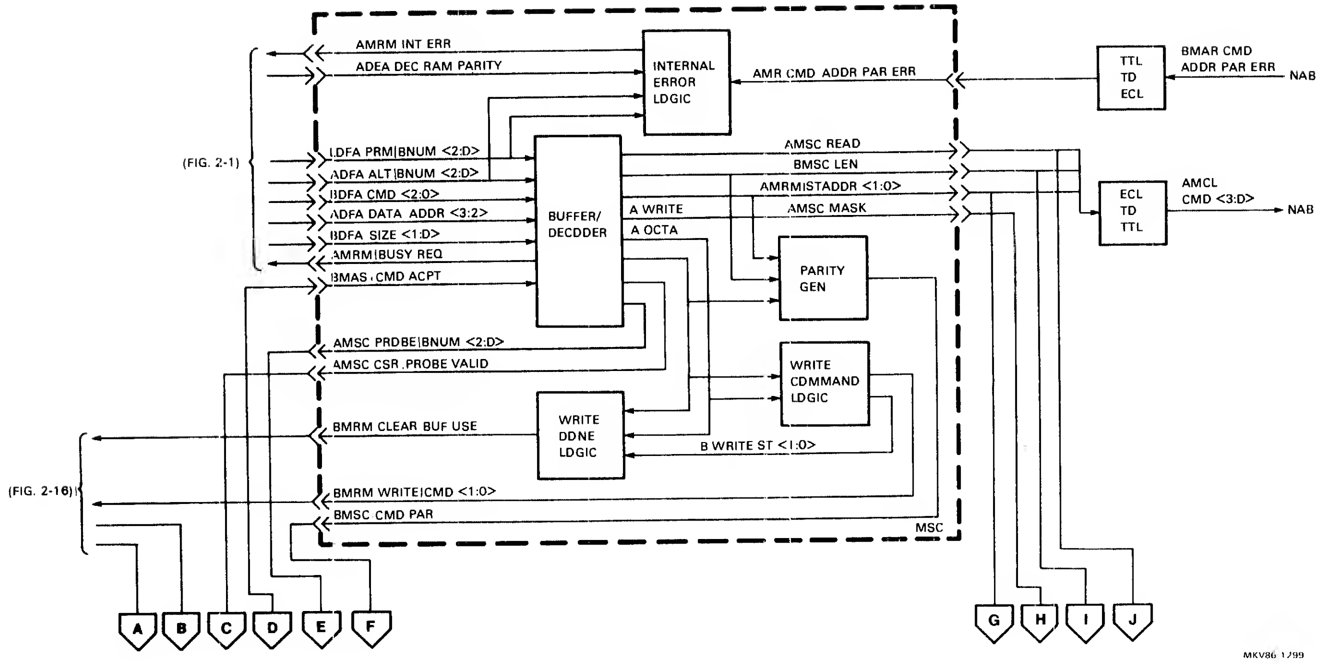


Figure 2-33 MRM Block Diagram (Sheet 1 of 4)

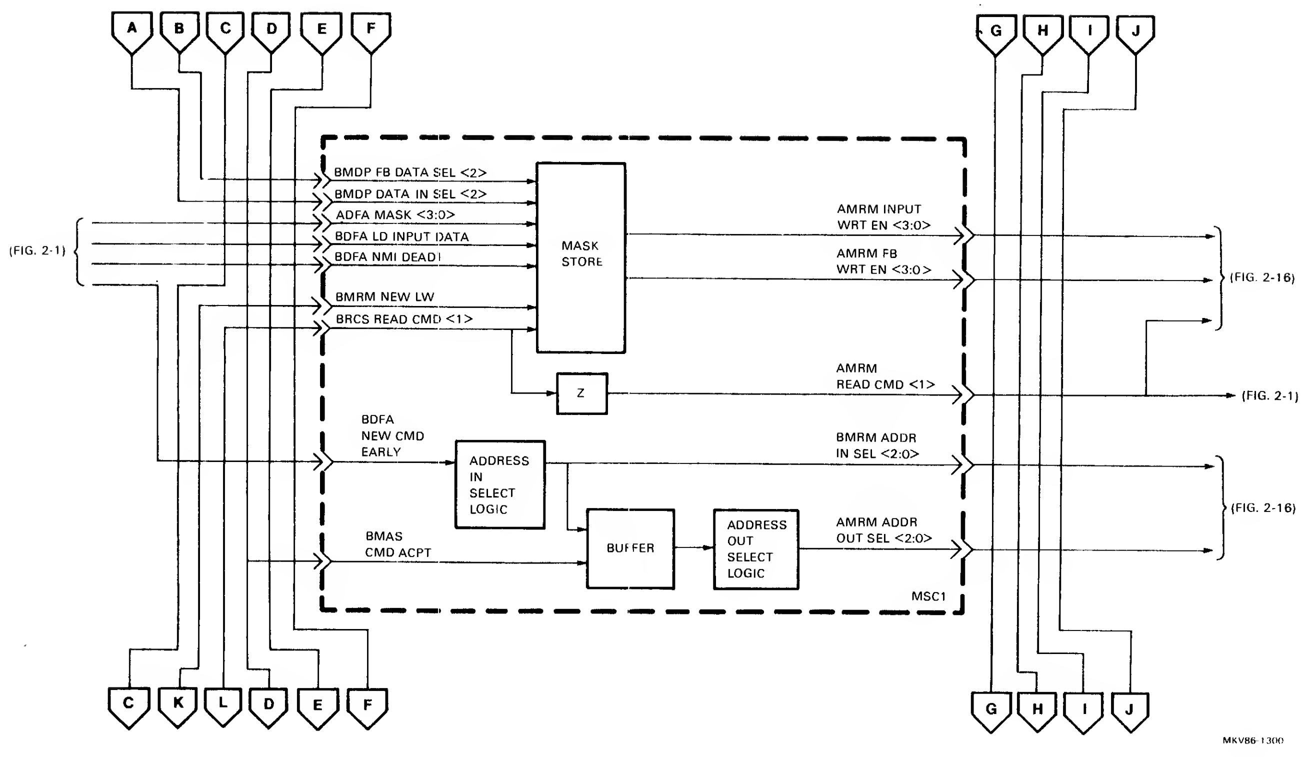


Figure 2-33 MRM Block Diagram (Sheet 2 of 4)

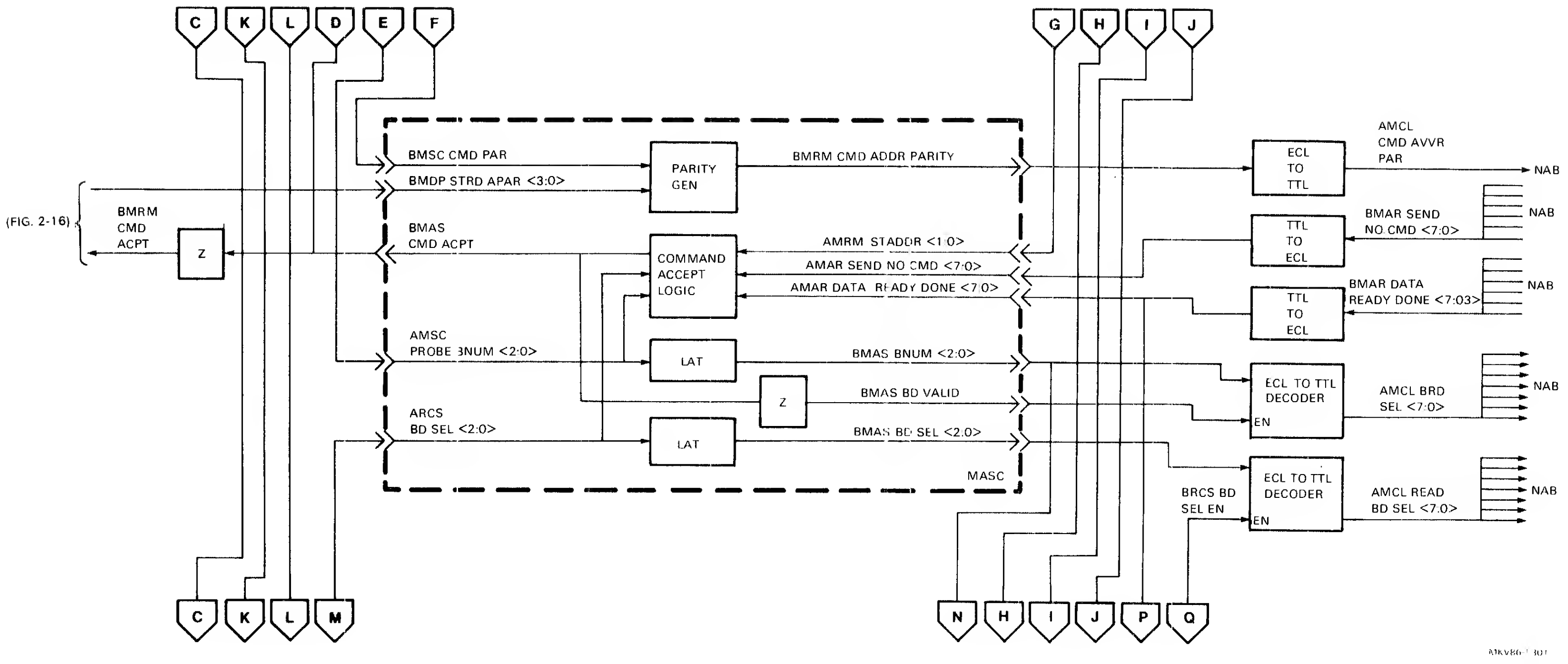


Figure 2-33 MRM Block Diagram (Sheet 3 of 4)

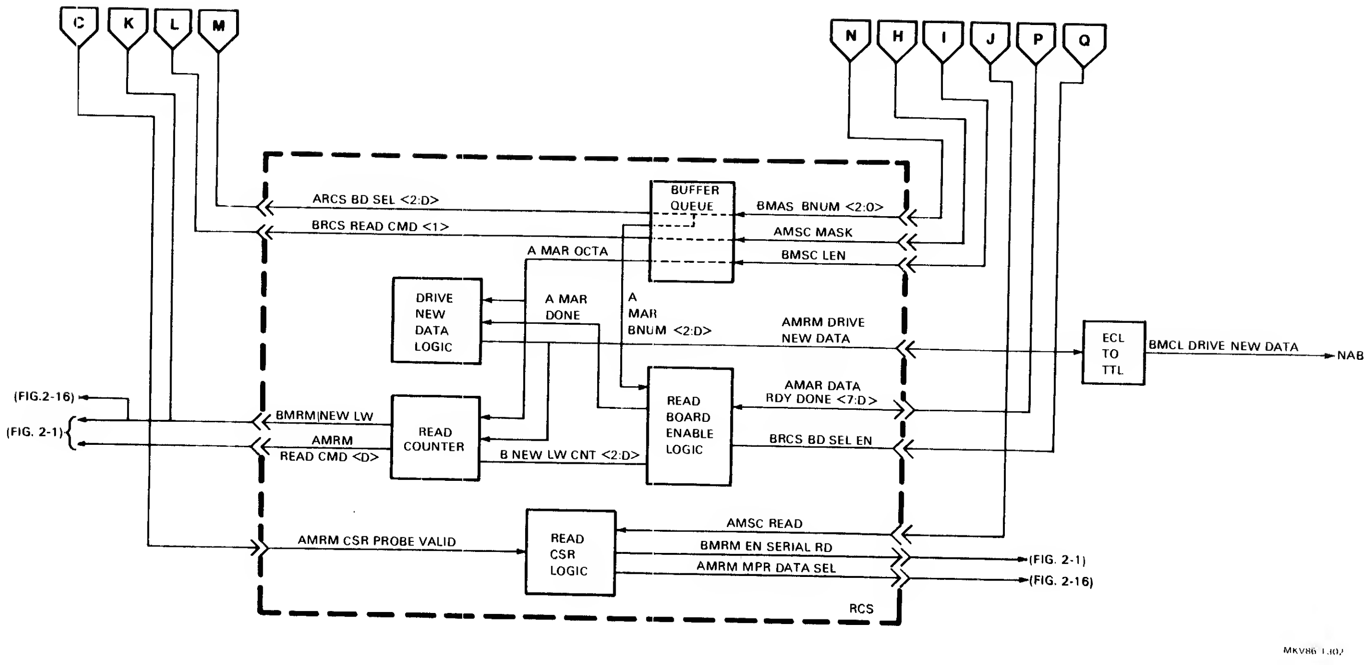


Figure 2-33 MRM Block Diagram (Sheet 4 of 4)

2.9.1 NAB Board Select

In the NAB command/address cycle, the MRM places onto the NAB the:

- Asserted board select line to the selected (X) array (AMCL BRD SEL<X>).
- Command field (AMCL CMD<3:0>).
- Command/address parity bit (AMCL CMD ADDR PAR).

Commands and board-select numbers are received from the DFA and are applied to a buffer/decoder in the MSC. Commands and board-select numbers for up to three commands can be held in the buffer until the array boards are ready to accept the commands. The buffer/decoder outputs command data from the first command it receives but will not output command data from the next command until it receives BMAS CMD ACPT from the MASC. The assertion of BMAS CMD ACPT indicates that the command data from the buffer/decoder has been accepted by the array board and the buffer/decoder can output the data from the next command. When the buffer/decoder has command data for two commands, it asserts AMRM BUSY REQ to the DFA to place memory-busy on the NMI. The third location in the buffer/decoder is filled only when a read command is received in the next bus cycle after the assertion of AMRM BUSY REQ.

The primary board-select number (ADFA PRM BNUM<2:0>) passes through the buffer/decoder and outputs the MSC as AMSC PROBE BNUM<2:0>. When the buffer/decoder senses the second octaword read of a hexword read operation, it uses the alternate board-select number (ADFA ALT BNUM<2:0>) for the AMSC PROBE BNUM<2:0> output.

AMSC PROBE BNUM<2:0> is applied to the MASC where it is latched to become BMAS BNUM<2:0>. AMSC PROBE BNUM<2:0> is also applied to command-accept logic which checks the send-no-command and data-ready-done status of the selected board. If the selected board can accept the command, the command-accept logic asserts BMAS CMD ACPT to inform the MCL of this fact. BMAS CMD ACPT becomes BMAS BD VALID which outputs the MASC and enables an ECL-to-TTL decoder. The decoder decodes the board select number from the MASC (BMAS BNUM<2:0>) to assert one of eight AMCL BRD SEL lines. The asserted AMCL BRD SEL line enables the selected array board to perform the commanded operation.

BMAS BNUM<2:0> is also applied to the RCS where the board number is stored in a buffer queue during read operations.

2.9.2 NAB Command Field and Parity

The four-bit NAB command field (AMCL CMD<3:0>; Figure 1-9) specifies the:

- Command function (read or write)
- Transfer (longword read/write or octaword read)
- Starting address (which of the four array banks is to be accessed).

The MRM functions to convert the other command types (masked writes, octaword writes, hexword reads) into the command types recognized by the array boards.

From the input command and size information, the buffer decoder outputs AMSC READ and BMSC LEN. AMSC READ specifies the command function. BMSC LEN specifies the size of the transfer. Address bits ADFA DATA ADDR<3:2> from the DFA specify starting address AMRM STADDR<1:0>.

A parity generator in the MSC generates a parity bit for the NAB command field which is sent to the MASC as BMSC CMD PAR. In the MASC the command parity bit is combined with the address parity bits from the MDP (BMDP STRD APAR<3:0>) to generate a command/address parity bit (BMRM CMD ADDR PARITY). The command/address parity bit is converted from ECL to TTL and placed on the NAB as AMCL CMD ADDR PAR.

The MSC buffer/decoder also outputs A WRITE and A OCTA to write-done logic. When the write-done logic senses a longword write operation, (A WRITE true and A OCTA false), it monitors the BMAS CMD ACPT signal from the MASC. When the array board has accepted the longword write command, BMAS CMD ACPT asserts causing the write-done logic to assert BMRM CLEAR BUF USE to the MDP. BMRM CLEAR BUF USE informs the MDB that it can place new data into the buffer that was just unloaded.

2.9.3 MDB Address Selection

Address-in select logic in the MSC1 generates select bits to locate the command address within the address area of the MDB. The select bits (BMRM ADDR SEL<2:0>) are sent to the MDP where they are applied to the W write port of the MDB. When a new command is received, the DFA FUNK asserts BDFA NEW CMD EARLY to the address-in select logic to increment the select bits to the next location in the MDB.

The address-in select bits are also applied through a buffer to address-out select logic where the select bits that locate the MDB output address are generated. The output select bits (AMRM ADDR OUT SEL<2:0>) are sent to the MDP where they are applied to the address read port of the MDB. The buffer can hold up to three sets of select bits. The first set of select bits are output from the buffer to the address-out logic. The assertion of BMAS CMD ACPT from the MASC (indicating that the array has accepted the current command address) causes the buffer to output the next set of select bits.

2.9.4 Internal Error

If the selected array board detects a parity error in the command field received from the MRM or in the address received from the MDP, it asserts BMAR CMD ADDR PAR ERR in the second NAB bus cycle after the command/address cycle (Figures 3-1 through 3-3). BMAR CMD ADDR PAR ERR enters the MSC as AMAR CMD ADDR PAR ERR where it is applied to internal-error logic. The internal-error logic generates parity on the primary and alternate board select numbers which is compared to a parity bit from the decode RAM (ADFA DEC RAM PARITY). If a parity error is detected in the decode RAM output or AMAR CMD ADDR PAR ERR is asserted, the internal-error logic asserts AMRM INT ERR to the DFA resulting in a memory-busy and a memory-interrupt being placed onto the NMI.

2.9.5 MDB Write-Data Load

During the write-data cycles, the MSC1 generates the load-enable signals that byte-load the write data from the NMI (via the DAD) into the MDB. The 4-bit load-enable signal (AMRM INPUT WRT EN<3:0>) is sent to the MDB where it is applied to the W write port. The load-enable signal is obtained from mask-store logic which asserts all four bits (to load the entire longword) in all cases except during a masked write when error correction is required on the read data (see Section 2.9.8).

2.9.6 Octaword Writes

For an octaword write operation, the MRM maintains the board select signal, the command field, and the command/address parity bit on the NAB to effect four longword writes of the memory array. A WRITE and A OCTA from the MSC buffer/decoder inform the write-done logic and the write-command logic of an octaword write operation while BMAS CMD ACPT informs them of each longword that has been accepted by the array board. The write-command logic generates write-command bits (BMRM WRITE CMD<1:0>) to the MDP to unload the write data from the MDB. It also supplies write-state bits (B WRITE ST<1:0>) to the write-done logic. The write-state bits are incremented by BMAS CMD ACPT. Thus, the write-done logic can determine when the fourth longword has been transferred to the arrays. When this occurs, the write-done logic asserts BMRM CLEAR BUF USE to the MDP to allow re-use of the MDB buffer that was just unloaded.

2.9.7 Read-Data Cycle(s)

In read operations (including reads associated with masked writes), the board select number (BMAS BNUM<2:0>) from the MASC along with AMSC MASK and BMSC LEN from the MSC, are entered into a three-stage buffer queue.

AMSC MASK outputs the RCS buffer-queue as read command bit <1> (BRCS READ CMD<1>) which specifies the read operation as masked or normal. BRCS READ CMD<1> is passed through the MSC1 and then sent to the DFA and MDB as AMRM READ CMD<1>.

The board number from the buffer queue outputs the RCS as ARCS BD SEL<2:0> and is applied to the MASC. In the MASC the board number is applied to the command-accept logic where it inhibits BMAS CMD ACPT for any command directed at the array board being read (AMSC PROBE BNUM<2:0> = ARCS BD SEL<2:0>). ARCS BD SEL<2:0> is also latched to become BMAS BD SEL<2:0> which is then applied to an ECL-to-TTL decoder. When enabled the ECL-to-TTL decoder asserts one of eight AMCL READ BD SEL lines to enable the array board to transfer its read data to the MCL. The decoder is enabled by BRCS BD SEL EN which is generated by the read-board enable logic in the RCS. The logic monitors the AMAR DATA RDY DONE line from the array board commanded to do a read as specified by A MAR BNUM<2:0>. When AMAR DATA RDY DONE asserts, the read-board enable logic asserts BRCS BD SEL EN to enable the ECL-to-TTL decoder which then enables the array board for a transfer of the read data by asserting AMCL READ BD SEL<X>.

The read-board select logic also asserts A MAR DONE to the drive-new data logic which then asserts AMRM DRIVE NEW DATA. AMRM DRIVE NEW DATA becomes BMCL DRIVE NEW DATA on the NAB which functions to clock the first read data longword from the array board.

AMAR DRIVE NEW DATA also inputs into a read counter which outputs BMRM NEW LW and AMRM READ CMD<0>. BMRM NEW LW informs the MCL that a longword of read data is coming. Read command bit <0> specifies the longword as the "first" longword transferred in the operation. If the drive-new-data logic senses this to be an octaword transfer (A MAR OCTA true) it asserts three more drive-new-data pulses to retrieve the other three longwords of read data from the array board. The read-counter, also sensing this to be an octaword transfer, asserts three more BMRM NEW LW pulses and specifies the three longwords to be "next" longwords via read-command bit <0>. During the transfer, the read counter outputs a longword count (B NEW LW CNT<2:0>) to the read-board-enable logic causing it to hold BRCS BD SEL EN asserted for the entire transfer.

2.9.8 Masked Writes

Insofar as the MRM is concerned, a masked write operation is basically a read of an array board followed by a write to the same board. The mask field (ADFA MASK<3:0>) is stored in MSC1 by BDFA LD INPUT DATA from the MDP. The mask fields associated with the data for two commands can be stored (corresponding to the data stored in the X and Y buffers of the MDB). The mask field is stored in an X or Y section according to the BMDP DATA IN SEL<2> bit from the MDP. (BMDP DATA IN SEL<2> specifies in which section of the MDB the write data is being stored).) When the mask-store senses a masked operation (via the BRCS READ CMD<1> bit) and that a longword of read data has been obtained from the array board (BMRM NEW LW asserts), it outputs the mask field as AMRM FB WRT EN<3:0> to byte load the read data into the MDB. The mask field is output from the X or Y section of mask store according to the BMDP FB DATA SEL<2> bit from the MDP. BMDP FB DATA SEL<2> specifies which section of the MDB (X buffer or Y buffer) is being unloaded.

If a single-bit error is detected in the read data, the mask-store is informed of this by the assertion of BDFA NMI DEAD1 from the DFA. The mask store then outputs the mask field as AMRM INPUT WRT EN<3:0> to byte-load the corrected read data into the MDB via the W write port. The corrected read data is loaded through the W write port because after being corrected in the DAD, it was wrapped around and returned to the MDB via the normal write data path.

2.9.9 CSR READS

When the read command is for a CSR read, the MSC buffer/decoder decodes this and outputs AMSC CSR PROBE VALID. AMSC CSR PROBE VALID is applied to read-CSR logic in the in the RCS to specify the read operation as a CSR read. The read-CSR logic responds by outputing BMRM EN SERIAL READ and AMRM MPR DATA SEL. BMEM EN SERIAL READ is sent to the DAD in the DFA where it switches the CSR data onto the data output path to the NMI. AMRM MPR DATA SEL is sent to the MDB where it switches a mux which wraps the write data output from the MDB back into the DFA as CSR feedback data. The wrapped write data is applied to the CSR logic in the DAD.

2.10 MEMORY SEQUENCE CONTROL (MSC) MCA

The MSC MCA (along with the MSC1 MCA) recives and processes commands for the array modules. Figure 2-34 is a block diagram of the MSC MCA. The block diagram divides the MSC into nine areas that perform the following functions.

- Buffer Control -- if a target array board is busy, the buffer control will queue up to three commands in the command/address/size buffer and the BNUM probe buffer. The buffer control asserts a busy request on the NMI when two commands are queued. When the array board is free, the buffer control allows the commands to pass through the buffers.

- BNUM Probe Buffer and Error Logic -- stores up to three board probe numbers. Checks for errors and asserts AMRM INT ERR to the MCL if an error is detected.

- Command/Address/Size Buffer -- stores command/address/size data associated with up to three commands.

- Size Logic -- generates size signals according to the size of the commanded transfer.

- Starting Address Logic -- generates starting address for array boards.

- Mask Address/Size Buffer -- stores address/size data associated with the read portions of up to three masked write operations.

- Write Command Logic -- generates write-command bits for unloading MDB. Indicates state of octaword write transfers.

- Mask Write Logic -- senses a masked write operation and generates appropriate control signals.

- Command Done Logic -- signifies completion of the MSC's involvement in normal and masked operations.

Figure 2-34 illustrates the major signals connecting the nine functional areas. Each functional area has a detailed block diagram that may show other signals not shown on the overall block diagram. These signals have their source or destination referenced, by figure number, to other functional area(s). Refer to Figure 2-34 throughout Section 2.10.

2.10.1 MSC Buffer Control (Figure 2-35)

2.10.1.1 **Buffer Control Operation** -- The assertion of BDFA NEW CMD LATE signifies a new valid command has entered the command/address/size buffer and the BNUM probe buffer. BDFA NEW CMD LATE is clocked through two latches to become B VALID CMD<1>.

If the array board has accepted the BNUM probe (BMAS CMD ACPT true from the MASC) and the MSC has completed processing the command (BMSC PRE CMD DONE true from the command-done logic), B CMD DONE asserts. B CMD DONE prevents B VALID CMD<1> from reaching the third buffer latch, hence, A VALID CMD<2> remains false supplying an asserted A FIRST signal to the command/address/size buffer and the BNUM probe buffer. A FIRST bypasses the commands around the latches in the command/address/size and the BNUM probe buffers.

If the array board is not able to accept the command or the MSC is still processing the command (as in the case of an octaword write or a hexword read), B CMD DONE will be false. In this case the third buffer latch is set and A VALID CMD<2> asserts. The assertion of A VALID CMD<2> will:

- Negate A FIRST to remove the bypass path in the command/address/size and the BNUM probe buffers.

- Assert B HOLD CS2 to the command/address/size and BNUM probe buffers to latch the command in the buffers.

- Latches A VALID CMD<2> to maintain B HOLD CS2 asserted.

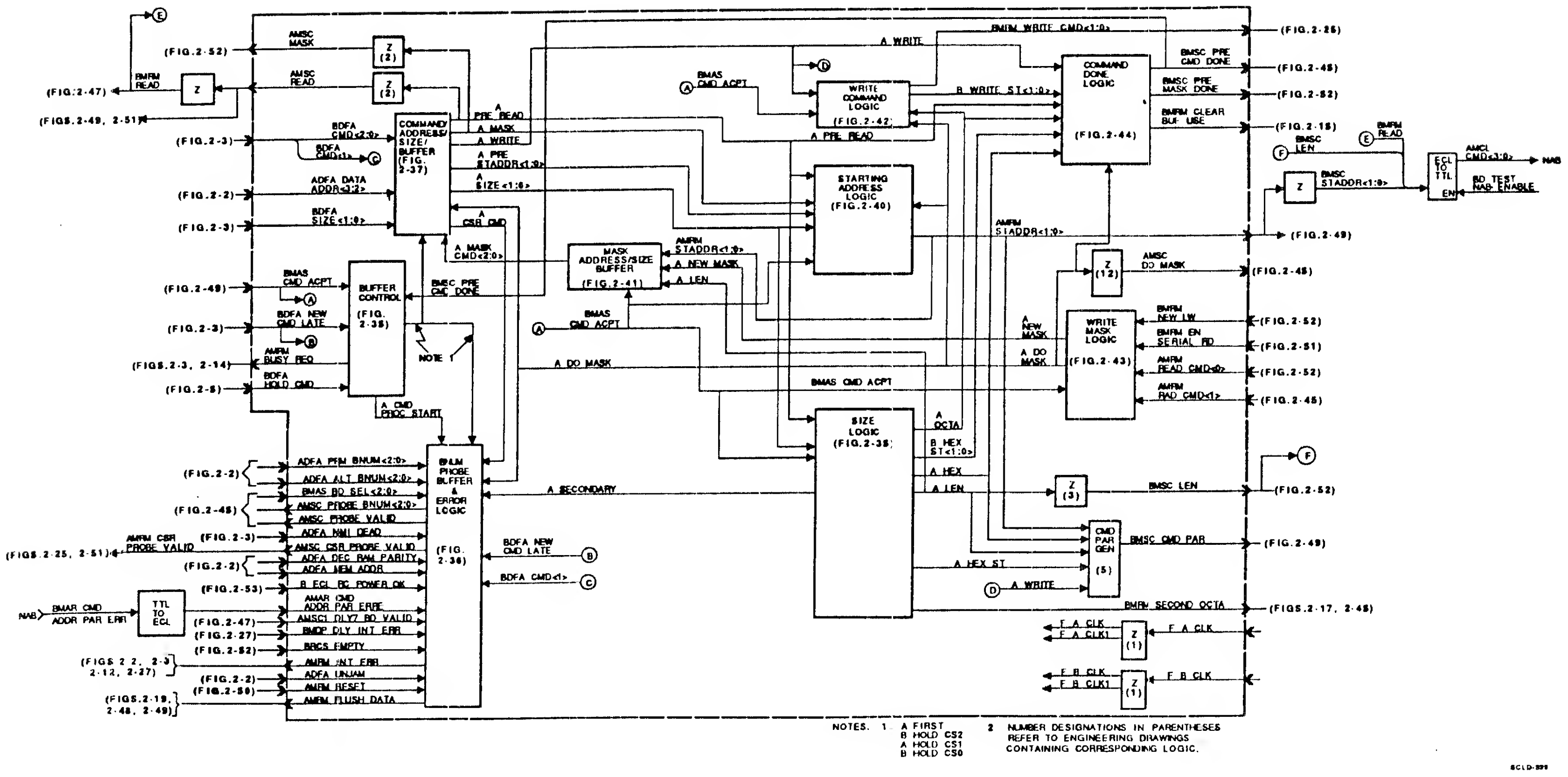
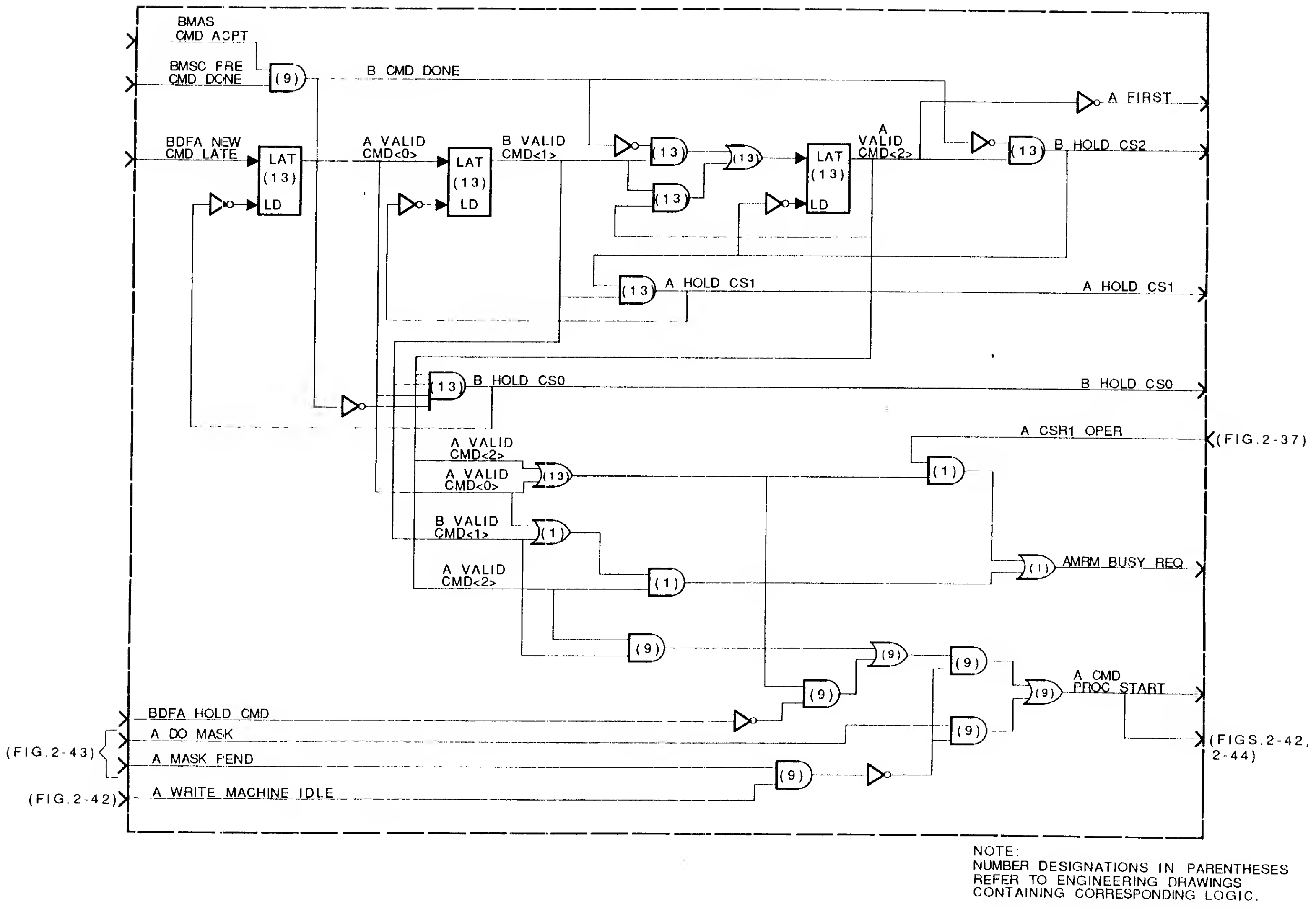


Figure 2-34 MSC Block Diagram



NOTE:
NUMBER DESIGNATIONS IN PARENTHESES REFER TO ENGINEERING DRAWINGS CONTAINING CORRESPONDING LOGIC.

SCLD 364

Figure 2-35 Buffer Control

B HOLD CS2 is applied to an A-HOLD-CS1 AND gate which looks for another command as indicated by the assertion of B VALID CMD<1>.

If another command is received, BDFA NEW CMD LATE asserts, followed by the assertion of A VALID CMD<0> and B VALID CMD<1>. B VALID CMD<1> enables the A-HOLD-CS1 AND gate asserting A HOLD CS1 to the two buffers. A HOLD CS1:

- Latches the second command in the two buffers.
- Latches B VALID CMD<1> to maintain A HOLD CS1 asserted.

A VALID CMD<2> is ANDed with either B VALID CMD<1> or A VALID CMD<0> to assert AMRM BUSY REQ. AMRM BUSY REQ causes memory-busy to assert on the NMI to stop new commands to memory. Only a read command received in the next bus cycle after the assertion of AMRM BUSY REQ will be accepted by the MCL.

A VALID CMD<2> and B VALID CMD<1> are applied to a B-HOLD-CS0 AND gate which looks for another command as indicated by the assertion of A VALID CMD<0>. If a read command occurs in the next bus cycle and the array board is still not free, BDFA NEW CMD LATE and A VALID CMD<0> asserts. The assertion of A VALID CMD<0> causes B HOLD CS0 to assert which:

- Latches the read command as the third command in the two buffers.

- Latches A VALID CMD<0> to maintain B HOLD CS0 asserted.

When the array board is free (BMAS CMD ACPT true) and the MSC has finished processing the current command* (BMSC PRE CMD DONE asserted), B CMD DONE asserts. B CMD DONE negates the three HOLD-CS latch signals and the commands output the command/address/size and BNUM probe buffers in order.

2.10.1.2 **AMRM BUSY REQ** -- When the buffer control has latched two commands in the buffers, it asserts AMRM BUSY REQ to the ARID which asserts memory-busy on the NMI. As seen in Figure 2-35, the true state of A VALID CMD<2> along with the the assertion of either A VALID CMD<0> or B VALID CMD<1>, will assert the busy request.

A CSR1 operation always requires that busy be asserted (see Section 2.2.6). The assertion of A CSR1 OPER from the command logic will assert the busy request when the CSR command is valid (A VALID CMD<2> if signals are queued; A VALID CMD<0> if no signals are queued).

* This would be four longword writes for an octaword write command or two octaword reads for a hexword read command.

2.10.1.3 A CMD PROC START -- A CMD PROC START is generated in the buffer control and sent to the BNUM probe buffer where it initiates the processing of a command by enabling the BNUM probe. A CMD PROC START is also sent to the write-command logic and the command-done logic to initiate command processing in those areas.

The write portion of a masked write has priority over other commands. A CMD PROC START is inhibited when the write portion of a masked write is pending (A MASK PEND true from the write-mask logic) and a multi-longword write operation is not already in progress (A WRITE MACHINE IDLE true from the write-command logic). When the write portion of the masked write is started, A DO MASK asserts (and A MASK PEND negates) from the write-mask logic. This results in the assertion of A CMD PROC START to start the write portion of the masked write. If the masked write is a quadword or octaword write, A CMD PROC START remains asserted for the duration of the transfer.

For a non-masked operation (and the read portion of a masked-write operation), A CMD PROC START asserts when:

- There is a valid command in the first or third location of the buffers (A VALID CMD<0> or A VALID CMD<2> true) while BDFA HOLD CMD is false. (The only time BDFA HOLD CMD is true is in single-step mode when a write command is being execute.d) Or,

- There is a valid command in the second and third locations of the command/address/size and BNUM probe buffers (B VALID CMD<1> and A VALID CMD<2> true).*

2.10.2 BNUM Probe Buffer and Error Logic (Figure 2-36)

2.10.2.1 Probe Logic -- The probe logic supplies the three-bit probe number (AMSC PROBE BNUM<2:0>) and a probe-valid signal (AMSC PROBE VALID) to the MASC. The probe number selects the array board to receive the current command. The probe-valid signal validates the probe number for the MASC.

Three-bit probe numbers ADFA PRM BNUM<2:0> and ADFA ALT BNUM<2:0> are recieved from the decode RAM and specify a primary and an alternate board number. The primary board number is used for longword, quadword, and octaword operations (only one array board is accessed). The alternate board number is used for the second octaword read of a hexword read operation. For interleaved oeration, the primary and alternate board numbers are different. For non-interleaved operation, they are the same.

* In this case, BDFA HOLD CMD is not a factor as A CMD PROC START is for the A VALID CMD<2> command whose data has already been loaded into the MDB.

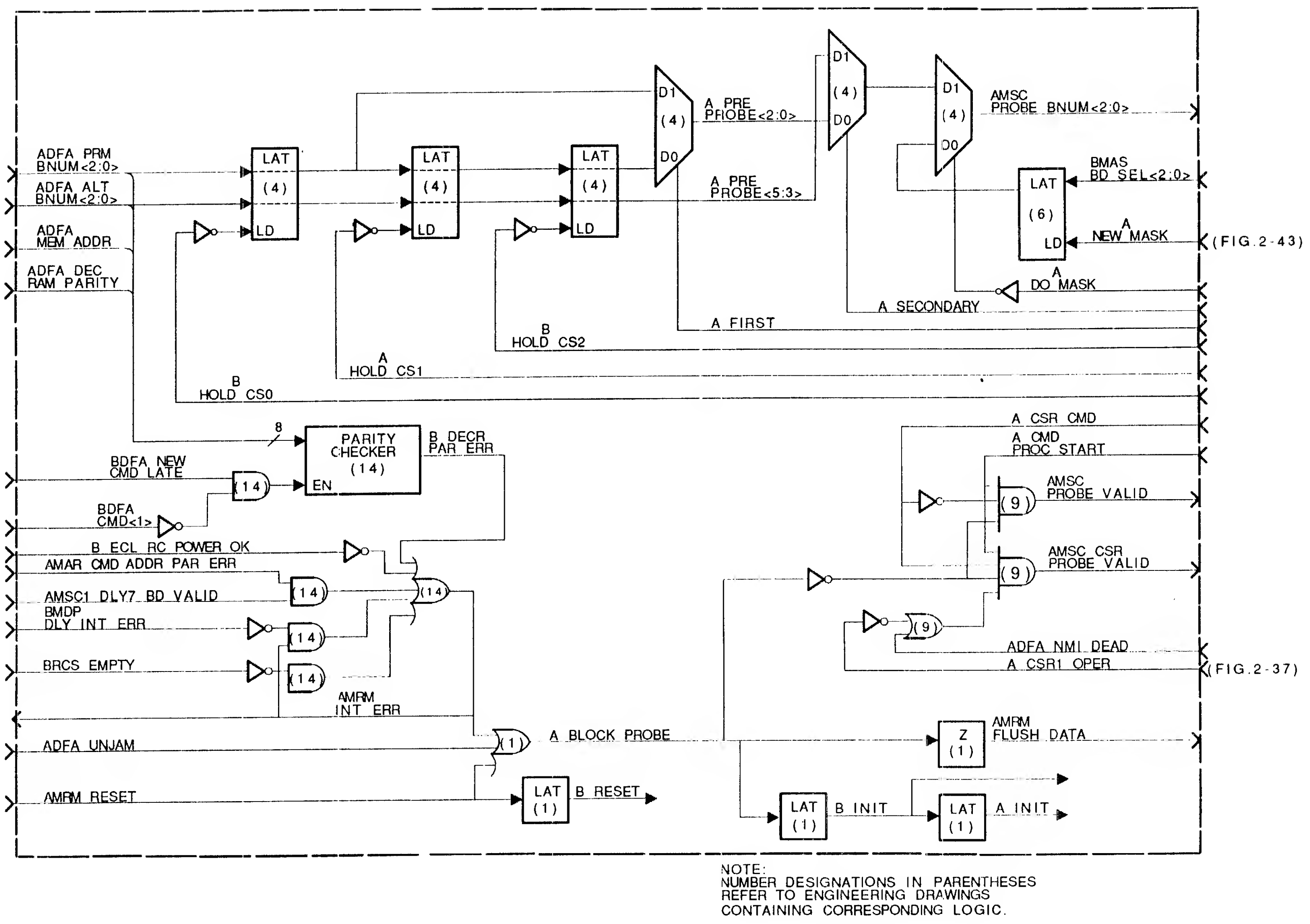


Figure 2-36 BNUM Probe Buffer and Error Logic

ADFA PRM BNUM<2:0> and ADFA ALT BNUM<2:0> are input into a three-stage buffer. If necessary, the buffer can hold the probe numbers associated with three queued commands. B HOLD CS2, A HOLD CS1, and B HOLD CS0 from the buffer control logic, latch the three probe numbers in the buffer. If no commands are queued, A FIRST from the buffer control logic is true and bypasses the primary probe number around the buffer latches via a mux.

The primary probe number output from the mux (A PRE PROBE<2:0>), is applied to another mux which also receives the alternate board number (A PRE PROBE<5:3>) from the buffer. This mux is controlled by A SECONDARY from the size logic. When the size logic detects the second octaword read of a hexword read, it asserts A SECONDARY to select the alternate probe number. Otherwise, the mux selects the primary board number.

The probe number output from the mux is applied to still another mux where it is selected during non-masked oprations and during the read portion of a masked-write operation. When the write portion of a masked-write is executing, A DO MASK is asserted from the mask-write logic causing the mux to select the BMAS BD SEL<2:0> probe number from the MASC. BMAS BD SEL<2:0> is the board number used by the MASC for the read portion of the masked-write. It was loaded into the BNUM probe buffer by A NEW MASK from the mask-write logic when the read portion was executing. It is now provided as the board number for the write portion.

A probe valid signal (AMSC PROBE VALID) is supplied to the MASC indicating that the probe number is valid. For the probe number to be valid:

- The buffer control logic must signal the start of a valid command process by asserting A CMD PROC START.
- This must be a memory access (A CSR CMD from the command/address/size buffer false).
- There must be no internal error, reset, or unjam conditions as indicated by the false state of A BLOCK PROBE.

If the current operation is a CSR access (not to the memory arrays), A CSR CMD will assert to inhibit a memory probe-valid signal, and assert a CSR probe-valid signal (AMSC CSR PROBE VALID). In addition, if this is a CSR1 access, it is required that ADFA NMI DEAD be true before AMSC CSR PROBE VALID assert. ADFA NMI DEAD indicates that the DAD data path is clear for accessing the decode RAM as required for a read or write of CSR1 (see Section 2.3.11).

AMSC CSR PROBE VALID is applied to the MDBC where it disables the write-command logic thereby disabling the MDB data-out function of the MDBC.

2.10.2.2 Error Logic -- The primary and alternate board numbers are applied to a parity checker along with two other decode RAM outputs; ADFA MEM ADDR (true when the memory is being addressed) and ADFA DEC RAM PARITY. If a parity error is detected, the checker asserts B DECR PAR ERR which in turn asserts AMRM INT ERR via an OR gate. AMRM INT ERR is distributed throughout the MRM as shown in Figure 2-34.

Other OR gate inputs that will cause an internal error are:

- The negation of B ECL RC POWER OK from the BBU.
- A command/address parity error from the memory array (AMAR CMD ADDR PAR ERR) if the array module is validated by AMSC1 DLY7 BD VALID from the MSC1.

AMRM INT ERR is locked-up and remains asserted for a minimum of 15 1/2 bus cycles after which BMDP DLY INT ERR is received from the MDBC to clear AMRM INT ERR. Fifteen and one-half bus cycles is sufficient time for the system to flush queues and prepare to start over. The flush of read commands queued in the RCS is checked by monitoring BRCS EMPTY. AMRM INT ERR will not clear until the RCS buffer is empty (BRCS EMPTY asserted).

AMRM INT ERR also asserts A BLOCK PROBE which prevents any command from executing by inhibiting any memory or CSR probe-valid signals. In addition, it asserts AMRM FLUSH DATA to the MDBC where it clears all the MDB input commands and resets the MDB input logic.

A system unjam or reset signal also asserts A BLOCK PROBE.

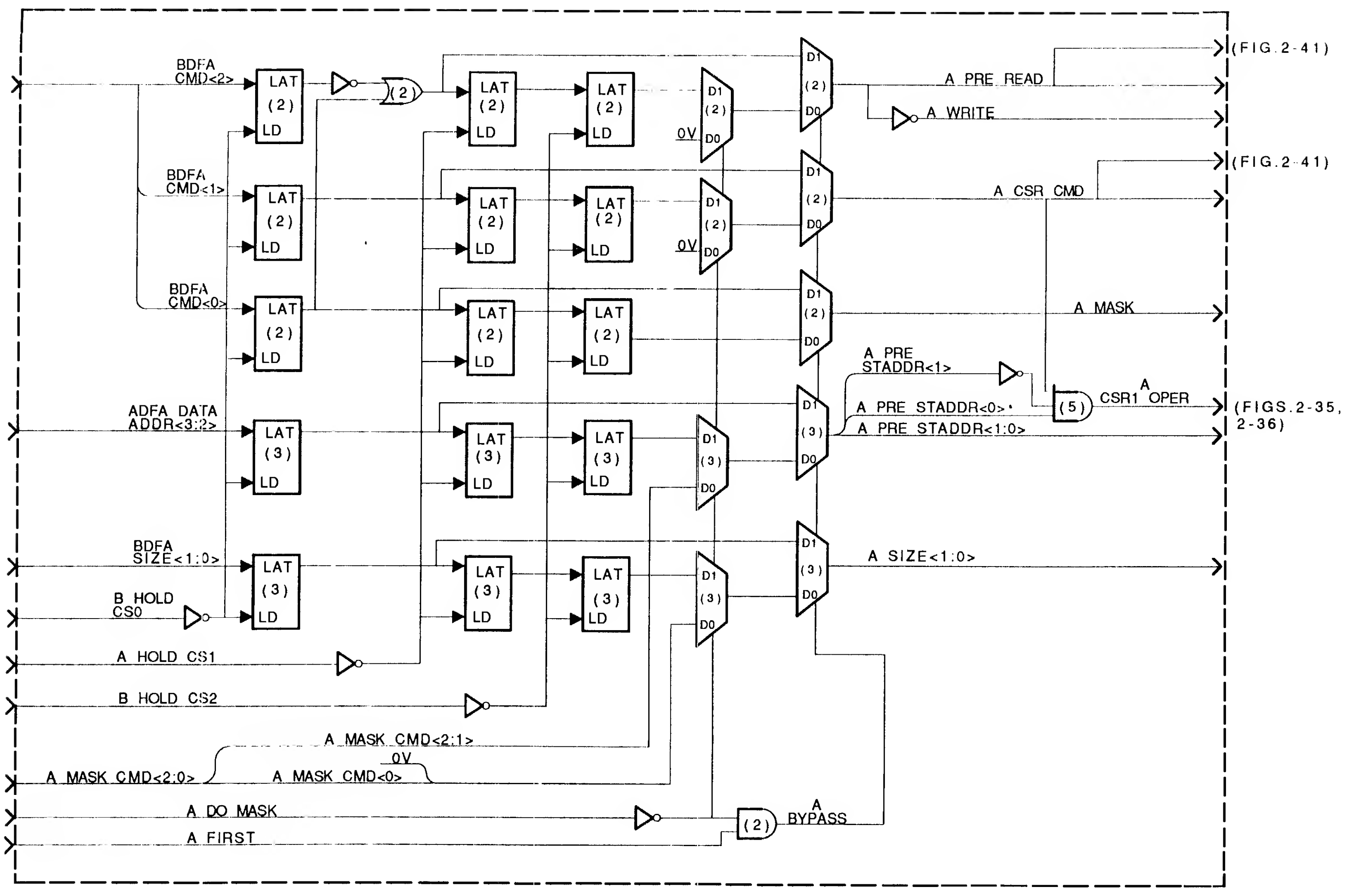
2.10.3 Command/Address/Size Buffer (Figure 2-37)

A three-bit command (BDFA CMD<2:0>) and a two-bit size code (BDFA SIZE<1:0>) from the FUNK, and a two-bit address (ADFA DATA ADDR<3:2>) from the DAD, are input into a three-stage buffer. If necessary, the buffer can hold the command, address, and size associated with three queued commands. B HOLD CS2, A HOLD CS1, and B HOLD CS0 from the buffer control logic, latch the command/address/size data in the buffer.

If no commands are queued, A FIRST from the buffer control logic is true and bypasses the command/address/size data around the buffer latches via a mux (except during the write portion of a masked-write).

2.10.3.1 Command Channel -- The three command bits are:

- BDFA CMD<2> = write bit; 0 = read; 1 = write
- BDFA CMD<1> = CSR bit; 0 = memory; 1 = CSR
- BDFA CMD<0> = mask bit; 0 = non-masked; 1 = masked



NOTE:
NUMBER DESIGNATIONS IN PARENTHESES
REFER TO ENGINEERING DRAWINGS
CONTAINING CORRESPONDING LOGIC.

SCLD-355

Figure 2-37 Command/Address/Size Buffer

The output of the read/write channel (A PRE READ) is asserted for a read operation. An inversion after the first latch accomplishes this. Note that in a masked write operation (BDFA CMD<0> asserted), A PRE READ is forced true for the read portion of the masked write.

When the command is a CSR access, A CSR CMD asserts. If the access is to CSR1, the address bit code will be 0:1 causing A CSR1 OPER to assert to the buffer control and the BNUM probe logic.

The read/write channel and the CSR channel contain a mux that feeds the output mux. During the write portion of a masked write, A DO MASK asserts and forces a 0 into the two channels via the muxes. This negates A PRE READ and A CSR CMD for the write operation.

2.10.3.2 Address/Size Channel -- The address bits output the address channel as A PRE STADDR<1:0>. The size bits output the size channel as A SIZE<1:0>.

During the write portion of a masked write, A DO MASK asserts and substitutes A MASK CMD<2:1> for the address bits, and 0 and A MASK CMD<0> for the size bits. A MASK CMD<2:0> are obtained from the mask address/size buffer where they were stored during the read portion of the masked write. A MASK CMD<2:0> are the starting address bits and the length bit associated with the read portion of the masked write and are now used as the starting address bits and the length bit for the write portion of the masked write.

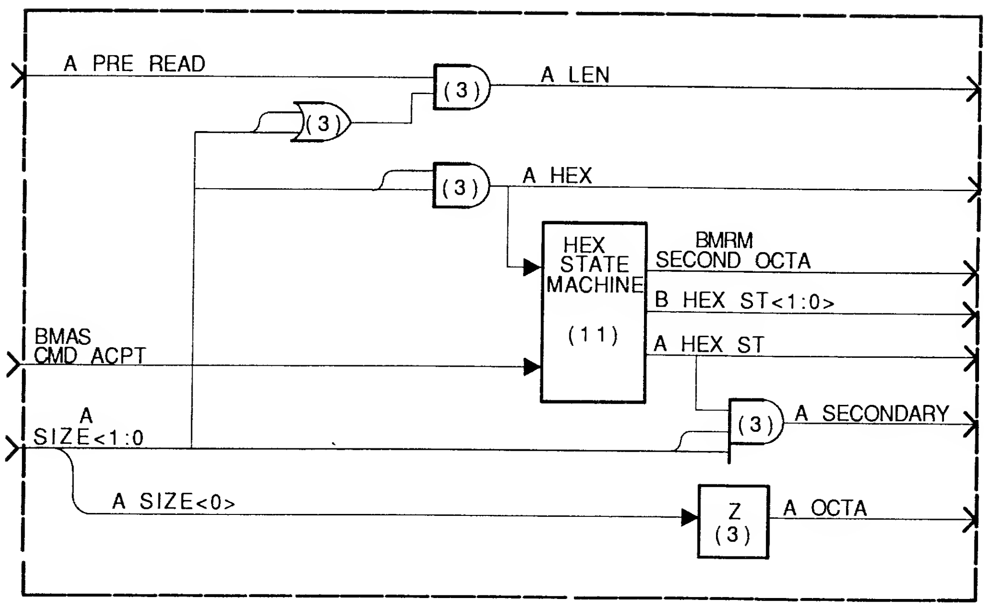
In addition, A DO MASK forces A BYPASS false so that the read/write and CSR channels will output the selected 0s, and the address and size channels will output the mask address and size command (A MASK CMD<2:0>).

2.10.4 Size Logic (Figure 2-38)

The size logic receives a two-bit size code (A SIZE<1:0>) from the command/address/size buffer, and generates signals specifying the size of the commanded transfer. The logic also contains a hex state machine that generates signals relating to a hexword read operation. The A SIZE<1:0> code is shown in Table 2-9.

Table 2-9 Size Code

A SIZE <1 0>	Size
0 0	Longword
0 1	Octaword
1 0	Quadword
1 1	Hexword



NOTE:
NUMBER DESIGNATIONS IN PARENTHESES
REFER TO ENGINEERING DRAWINGS
CONTAINING CORRESPONDING LOGIC.

SCLD-356

Figure 2-38 Size Logic

In discussing the size logic, three points must be noted:

- A quadword transfer is treated as an octaword transfer with no data transferred for the third and fourth octaword locations.
- A hexword transfer is treated as two octaword transfers.
- The array boards recognize only two size of transfers: a longword transfer (read or write) and an octaword transfer (read only)

A LEN is the length bit sent to the array board as part of the array command. It is negated for a longword transfer and asserted for an octaword transfer. A LEN is obtained from an AND gate that receives A PRE READ from the command/address/size buffer. For a write operation, A PRE READ is false making A LEN false to specify a longword transfer. For a read operation (A PRE READ true), the size bits are examined and will assert A LEN except when a longword is specified (size = 0:0).

A OCTA indicates that the current command is an octaword operation. A OCTA is identical to A SIZE<0> which is asserted for an octaword transfer and a hexword transfer (two octaword transfers).

A HEX indicates that the current command is a hexword operation. A HEX asserts when a size code of 1:1 is detected.

A hex state machine receives A HEX and BMAS CMD ACPT which places the machine into various hex states during the execution of a hexword read operation. The machine generates a two-bit hex state code (B HEX ST<1:0>) that indicates the current state of the machine.

Figure 2-39 is a flow diagram of the operation of the hex state machine during the execution of a hexword command. The machine idles in state 0 while waiting for a hex command. When a hex command is received, A HEX asserts causing the machine to advance to state 2 (1:0) while the MCL attempts to send the first octaword read command. When the array board accepts the octaword command, BMAS CMD ACPT asserts which in turn asserts A HEX ST. A HEX ST signifies that the first octaword has been sent.

A HEX will still be asserted to send the second octaword command, unless the write portion of a masked write is ready to execute. If this is the case, A HEX will be false placing the machine into state 3 (1:1) until the masked write completes. While A HEX is false, the state machine does not respond to BMAS CMD ACPT (which will assert due to the masked write).

The true state of A HEX places the machine into state 1 (0:1) signifying that the MCL is attempting to send the second octaword command. When in state 1, the machine outputs BMRM SECOMD OCTA signifying that the current command is the second octaword of a hexword transfer. BMAS CMD ACPT asserts when the second octaword has been accepted by the array board. The state machine then returns to state 0 to await another hexword read command.

BMRM SECOND OCTA is used to increment the address from the MDB for the second octaword location.

A HEX ST is applied to an AND gate that checks for a size code of 1:1. When a 1:1 size code is detected (hex transfer), A SECONDARY is asserted to the BNUM probe buffer to select the alternate board number for the second octaword transfer.

2.10.5 Starting Address Logic (Figure 2-40)

The starting address logic converts the pre-starting address received from the command/address/size buffer (A PRE STADDR<1:0>) into an initial starting address (A INIT STADDR<1:0>) that specifies which bank on the array board is to be accessed first. In addition, the logic increments the starting address during an octaword write operation to access the four banks on the array board.

2.10.5.1 Initial Starting Address -- The pre-starting address is passed through the initial address logic to become the initial

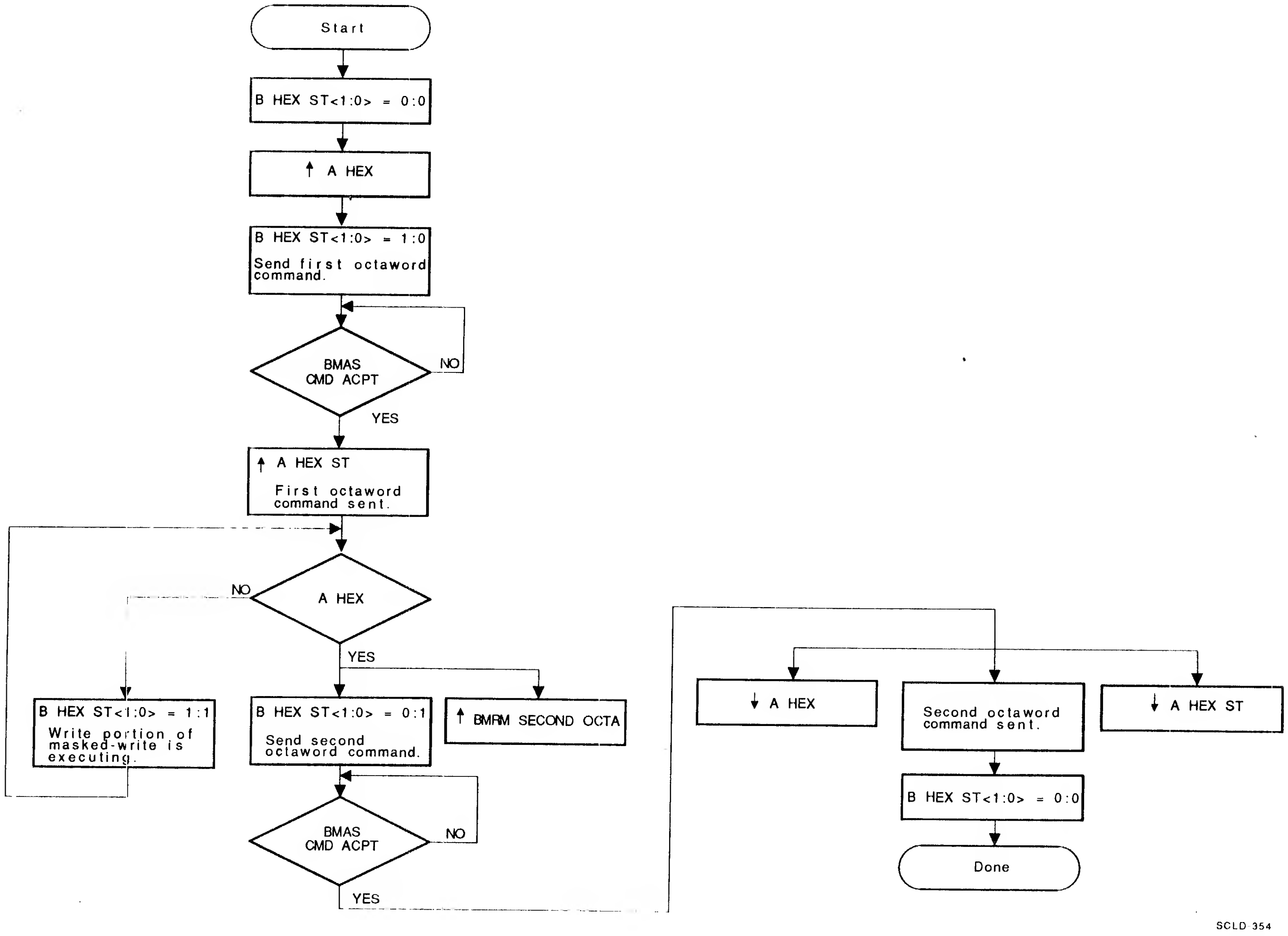
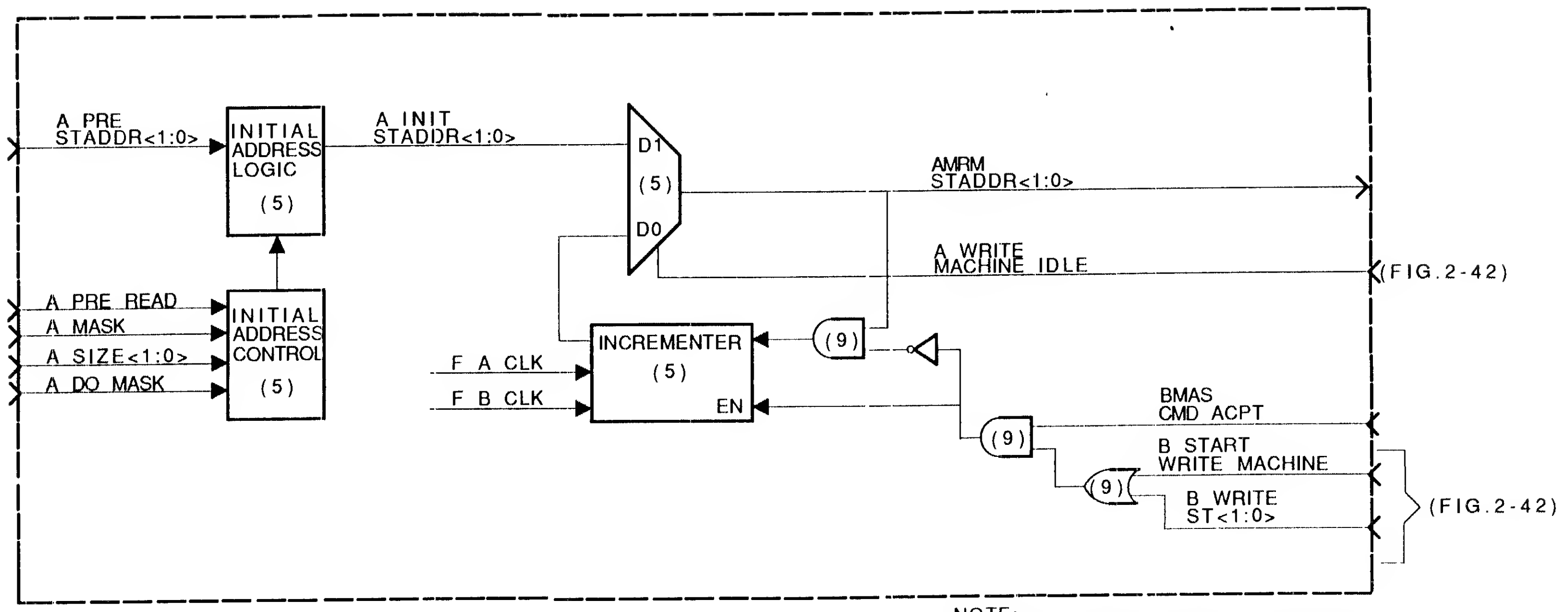


Figure 2-39 Hex State Machine Flow Diagram



NOTE:
NUMBER DESIGNATIONS IN PARENTHESES REFER TO ENGINEERING DRAWINGS CONTAINING CORRESPONDING LOGIC.

SCLD-361

Figure 2-40 Starting Address Logic

address. An initial address control forces specific pre-address bits to zero in the initial address logic, thereby restricting the choice of banks that can be selected as the starting bank. Table 2-10 is a truth table of the initial address.

Table 2-10 Initial Address Truth Table

Command	A INIT STADDR <1 : 0>*		Bank Accessed First
All Reads	X	X	Any
LW Writes	X	X	Any
QW Writes	X	0	First or third
OW Writes	0	0	First

* Bits shown as 0 are forced.
X = Corresponding A PRE STADDR<1:0> bit.

For all normal reads, the initial address is identical to the pre-address. A longword read can access any bank and an octaword read can start at any bank. The initial address control checks for an asserted A PRE READ and a negated A MASK to establish the command as a normal read.

For normal or masked longword writes, the initial address is identical to the pre-address. A normal or masked longword write can be made to any bank. The initial address control checks for a negated A PRE READ and a longword size code (0:0) to establish the command as a longword write.

For a masked quadword write, pre-address starting bit <0> is forced zero. A masked quadword write must start with either the first or the third array bank. The initial address control checks for a negated A PRE READ and a quadword size code (1:0) to establish the command as a quadword write.

For normal or masked octaword writes, both pre-address bits are forced zero giving an initial starting address of 0:0. A normal or masked octaword write must start with the first array bank (bank 0). The initial address control checks for a negated A PRE READ and an octaword size code (0:1) to establish the command as an octaword write.

The initial address control receives an asserted A DO MASK during the write portion of a masked write operation, which tells the control to use the pre-starting address unchanged. During the write portion of a masked-write, the pre-starting address is the initial starting address used for the read portion of the masked write that has been stored in the mask address/size buffer.

2.10.5.2 Address Incrementation -- The initial address from the initial address logic (A INIT STADDR<1:0>) is passed through a mux and output from the MSC as AMRM STADDR<1:0> for transmission to the array board. The true state of A WRITE MACHINE IDLE from the write-command logic causes the mux to select the initial starting address as the AMRM STADDR<1:0> output.

For a longword read or write, the initial starting address specifies the only bank to be accessed.

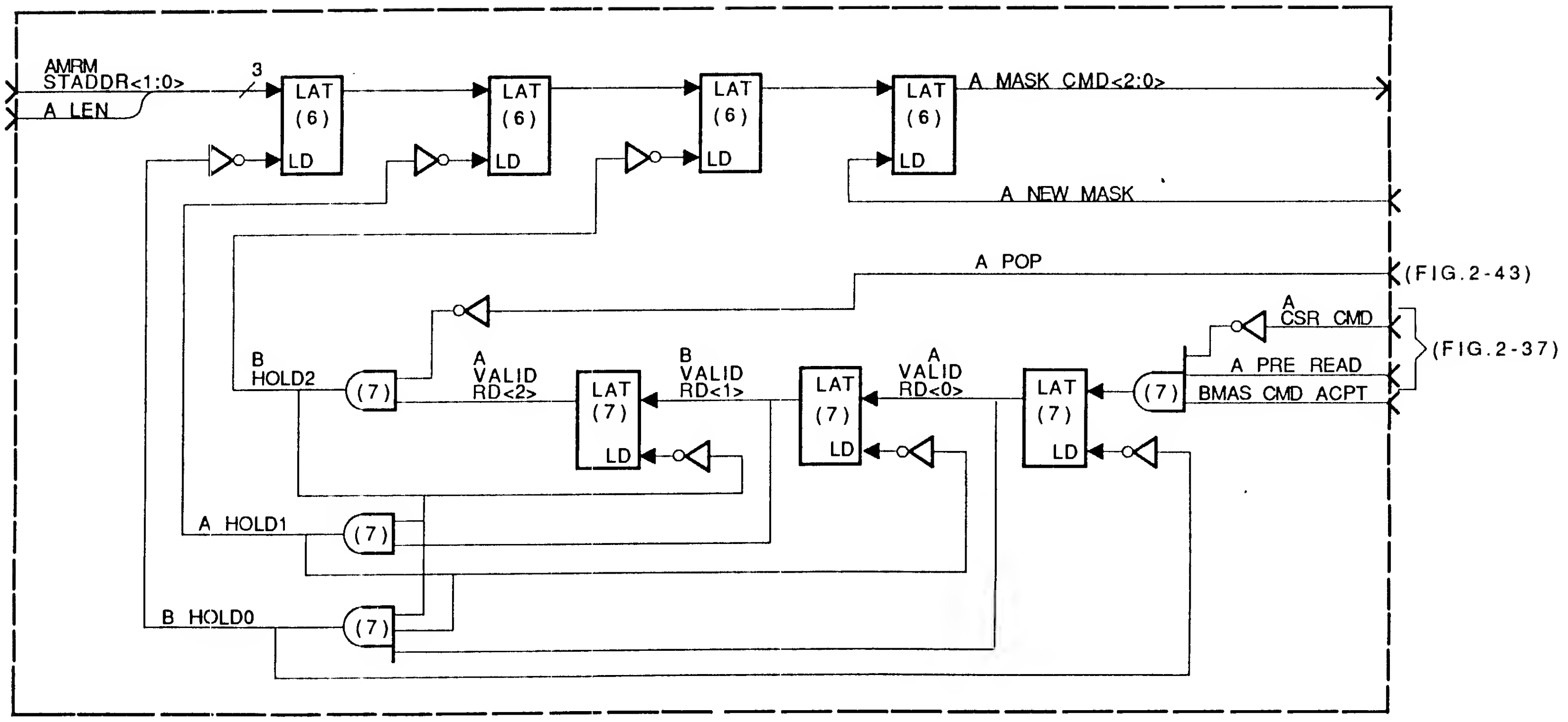
For an octaword read, the array board needs only the starting address. The length command bit specifies an octaword transfer to the array boards which reads the four array banks starting with the bank selected by the initial starting address.

For an octaword (and quadword) write, the initial starting address must be incremented because four longword writes are to be executed. The initial starting address (AMRM STADDR<1:0>) is loaded into an incrementer. The incrementer is enabled by B START WRITE MACHINE from the write-command logic when an octaword write command is received. B START WRITE MACHINE is validated by BMAS CMD ACPT when the array board has accepted the BNUM probe. When the incrementer is enabled, the initial starting address is removed, and F A CLK and F B CLK function to increment the initial address to select subsequent banks in the array board. Once the write machine starts, the write-command logic negates A WRITE MACHINE IDLE causing the mux to select the incremented address for the AMRM STADDR<1:0> output. The incrementer is held enabled for the octaword command by the binary state code (B WRITE ST<1:0>) from the write machine.

2.10.6 Mask Address/Size Buffer (Figure 2-41)

The mask address/size buffer receives the starting address (AMRM STADDR<1:0>) and the length bit (A LEN) associated with every command sent to the array boards. The starting address and length bit pass through three buffer stages and are loaded into an output latch by A NEW MASK from the mask-write logic. A NEW MASK asserts only when the mask-write logic senses the read portion of a masked write operation. Hence, only the starting address and length bit associated with a masked write are loaded into the output latch and made available to the command/address/size buffer.

The starting address and length bit associated with three read operations (including masked-write reads) can be stored in the buffer. When a read command is received (A PRE READ true), and the array board has accepted the command (BMAS CMD ACPT true), and the read access is to memory (A CSR CMD false), a latch is set asserting A VALID RD<0>. A VALID RD<0> becomes B VALID RD<1> which then becomes A VALID RD<2>. A POP is received from the masked-write logic and is asserted every time a new read command executes. If A POP is false when A VALID RD<2> asserts, the read command didn't execute (array board busy) and B HOLD2 asserts to latch the starting address and length bit associated with the



NOTE:
NUMBER DESIGNATIONS IN PARENTHESES REFER TO ENGINEERING DRAWINGS CONTAINING CORRESPONDING LOGIC.

SCLD-360

Figure 2-41 Mask Address/Size Buffer

pending read command. B HOLD2 latches A VALID RD<2> to maintain the B HOLD2 latch signal.

If another read command is received while A POP is still false, A VALID RD<0> asserts causing B VALID RD<1> to assert. B VALID RD<1> causes A HOLD1 to assert and latch the starting address and length bit associated with the second read command. A HOLD1 latches up B VALID RD<1> to maintain the A HOLD1 latch signal.

Likewise, if a third read command is accepted, it will assert A VALID RD<0> and B HOLD0 to latch up the third set of address and length bits.

When A POP does assert, B HOLD2 negates, in turn negating A HOLD1 and B HOLD0, allowing the address and length bits to proceed through the buffer. Address and length bits that are part of a masked write will get loaded into the output latch by A MASK and be applied to the command/address/size buffer as A MASK CMD<2:0>.

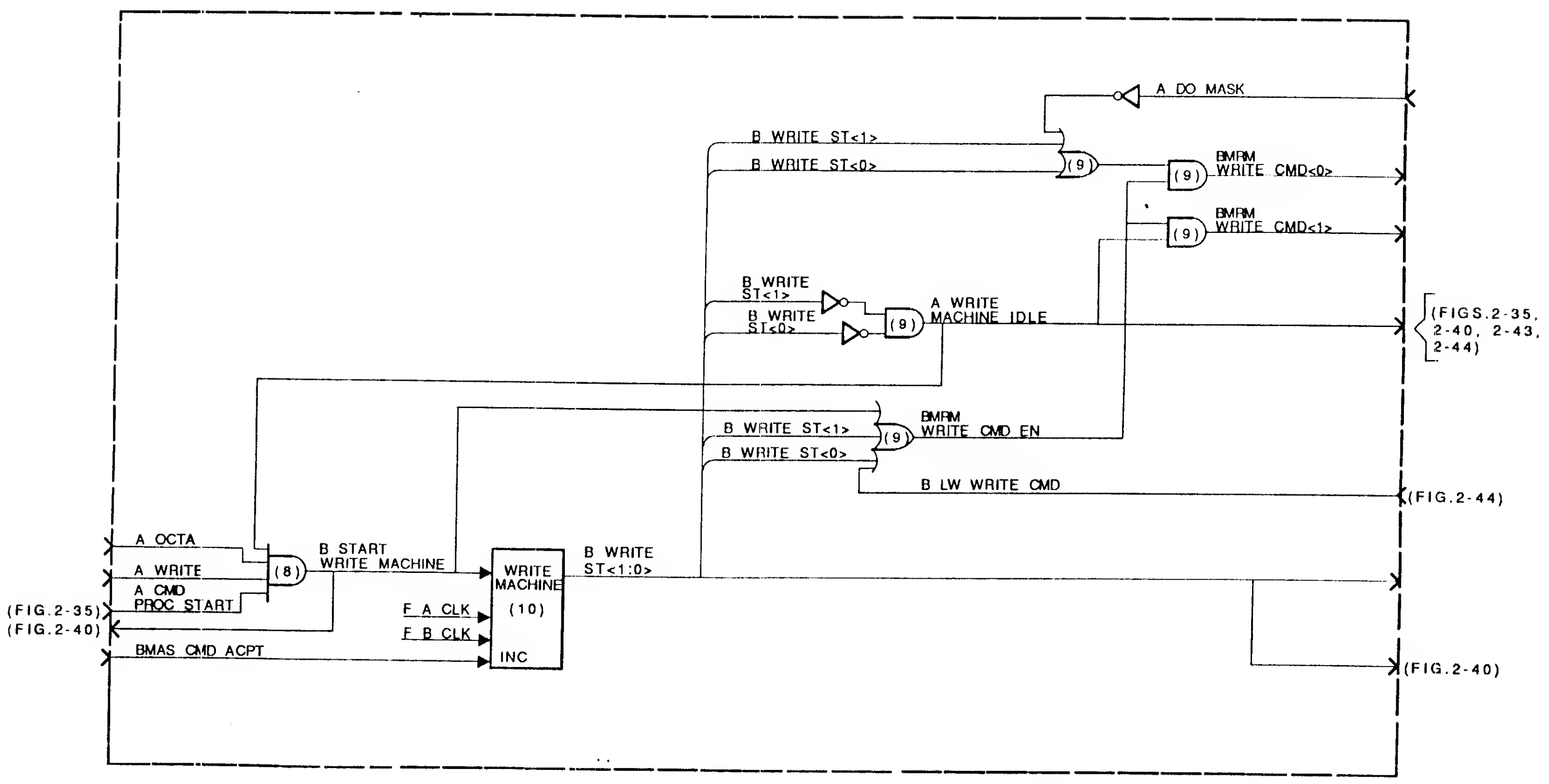
2.10.7 Write Command Logic (Figure 2-42)

The write command logic generates a two-bit write command (BMRM WRITE CMD<1:0>) for the MDBC that specifies a write command as the first longword of a transfer or a "next" longword, and identifies the first longword as masked or normal. The logic also contains a write-state machine that generates a two-bit, write-state code (B WRITE ST<1:0>) related to an octaword write operation.

2.10.7.1 Write Machine -- The write machine is started by the assertion of B START WRITE MACHINE when an octaword write command is received. B START WRITE MACHINE asserts when:

- The write machine is in its neutral state (A WRITE MACHINE IDLE true).
- The buffer control has allowed a command process to start (A CMD PROC START true).
- The command is a write (A WRITE true).
- The command is an octaword transfer (A OCTA true).

Once the write machine is started, the write-state code is incremented by BMAS CMD ACPT each time the array board accepts a longword write command. The write-state code indicates the status of the octaword transfer by indicating which longword is being transferred. The write-state code does not increment in binary format, as can be seen in Table 2-11.



NOTE:
NUMBER DESIGNATIONS IN PARENTHESES REFER TO ENGINEERING DRAWINGS CONTAINING CORRESPONDING LOGIC.

SCLD-358

Figure 2-42 Write Command Logic

Table 2-11 Write State Code

Longword Being Transferred	B WRITE ST <1 : 0>		Binary State
1st longword	0	0	0
2nd longword	0	1	1
3rd longword	1	1	3
4th longword	1	0	2

Once the write state code advances from its neutral (0:0) state, A WRITE MACHINE IDLE negates which in turn negates B START WRITE MACHINE.

2.10.7.2 Write Command Bits -- Write command bits BMRM WRITE CMD<1:0> identify every write command as the first or following longword in a write transfer, and the first longword as masked or normal. The bits are generated by two AND gates enabled by BMRM WRITE CMD EN whenever a write command is issued by the MRM. BMRM WRITE CMD EN is asserted by:

- B LW WRITE CMD from the command-done logic whenever a longword write command is received, or by
- B START WRITE MACHINE for the first longword transferred in an octaword write command, or by
- The B WRITE ST<1:0> code during the 2nd, 3rd, and 4th longword transferred during an octaword write command.

Table 2-12 defines the write command bit code that is generated.

Table 2-12 Write Command Code

BMRM WRITE CMD <1	0>	Function
0	0	No Op
0	1	Next longword (normal or masked)
1	0	First masked longword
1	1	First normal longword

As seen in Table 2-12, an asserted BMRM WRITE CMD<1> indicates the first longword in a transfer. BMRM WRITE CMD<1> is asserted by A WRITE MACHINE IDLE which is true except when the write machine is stepping through the second, third, or fourth longword of an octaword transfer.

An asserted BMRM WRITE CMD<0> performs two functions:

- It indicates the first longword in a transfer as being normal.
- It indicates a longword as being a "next" longword.

During the first longword of a write transfer, the write state-code is 0:0 making BMRM WRITE CMD<0> a function of A DO MASK. A DO MASK is true for a masked write causing BMRM WRITE CMD<0> to negate. For normal writes, A DO MASK is false and BMRM WRITE CMD<0> is asserted.

During the second, third, and fourth longword of an octaword transfer, the write-state code has at least one bit asserted, thereby keeping BMRM WRITE CMD<0> asserted to indicate the three longwords as "next" longwords.

2.10.8 Masked-Write Logic (Figure 2-43)

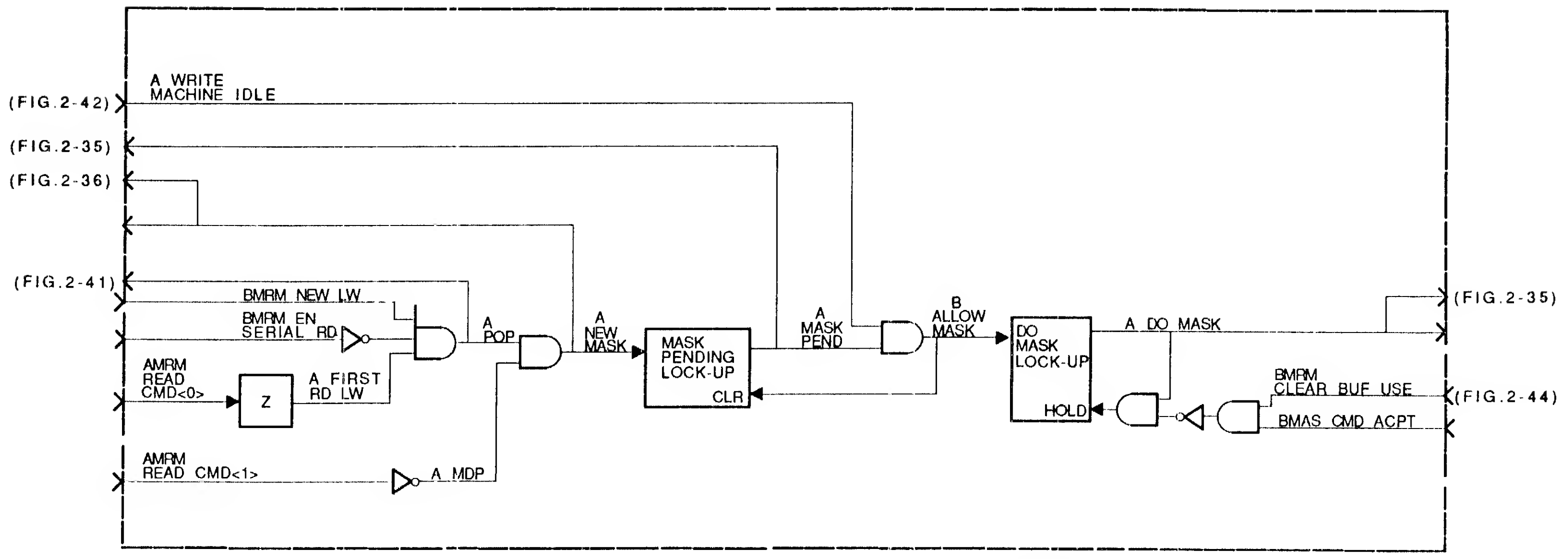
The masked-write logic senses when the write portion of a masked write is commanded and generates A NEW MASK and A DO MASK to inform the MRM.

The masked-write logic looks at read command bits AMRM READ CMD<1:0> from the RCS and the MSC1 to determine if a read command is normal or masked, and if the current longword is the first longword in the transfer or a "next" longword (see Table 2-6).

AMRM READ CMD<0> is true for the first longword of a read transfer and asserts A FIRST RD LW. If the read bits are validated by BMRM NEW LW, and this is not a CSR read (BMRM EN SERIAL RD false), A POP asserts. A POP is sent to the mask address/size buffer where it allows any queued address/length command to pass through the buffer. Hence, A POP asserts for the first longword of all read transfers (masked and normal).

If the read command is part of a masked write operation (AMRM READ CMD<1> false), A MDP is true and asserts A NEW MASK to the BNUM probe buffer and the mask address/size buffer specifying the first longword as being part of a masked transfer. In the BNUM probe buffer, A NEW MASK loads the array board number from the MASC. In the mask address/size buffer, A NEW MASK loads in the address/size data from the buffer queue making it available to the command/address/size buffer.

A NEW MASK is also applied to a mask-pending lock-up block which asserts and holds A MASK PEND. A MASK PEND is applied to the buffer control to inhibit a new command process from being initiated until the masked write is completed. In addition, A MASK PEND checks that the write machine is not executing an octaword operation (A WRITE MACHINE IDLE true) and then asserts B ALLOW MASK to a do-mask lock-up block. B ALLOW MASK clears A MASK PEND while the do-mask lock-up block asserts and holds A DO MASK. A DO



NOTE:
THE LOGIC IN THIS FIGURE IS CONTAINED ON SHEET 12 OF THE ENGINEERING DRAWINGS.

SCLD-357

Figure 2-43 Mask Write Logic

MASK stays true until the masked write is completed as indicated by the assertion of BMRM CLEAR BUF USE (validated by BMAS CMD ACPT) from the command-done logic.

2.10.9 Command Done Logic (Figure 2-44)

The command-done logic informs the MRM that the MSC has completed processing the current command and is driving command data onto the NAB. This is indicated by the assertion of BMSC PRE CMD DONE or BMSC PRE MASK DONE. BMSC PRE CMD DONE is asserted to the buffer control in the MSC and to the MSC1 for all read commands (including the read portion of a masked-write command) and all normal write commands. BMSC PRE MASK DONE is asserted to the RCS for all masked-write commands.

2.10.9.1 BMSC PRE CMD DONE --

A. Non-Hex Read Commands Done

The completion of MSC processing of a non-hex read command (A PRE READ true and A HEX false) is indicated by the assertion of B READ CMD. The assertion of B READ CMD requires that the command data be in the MSC buffers (A CMD PROC START true) and that the write machine be in its idle state (A WRITE MACHINE IDLE true).

The assertion of B READ CMD results in the assertion of BMSC PRE CMD DONE.

B. Hex Read Commands Done

The completion of MSC processing of a hexword read command is indicated by a hex state of 0:1 (B HEX ST<1:0> = 0:1) from the hex state machine in the size logic. A hex state of 0:1 indicates the second octaword read command of a hexword read has been sent to the array board*.

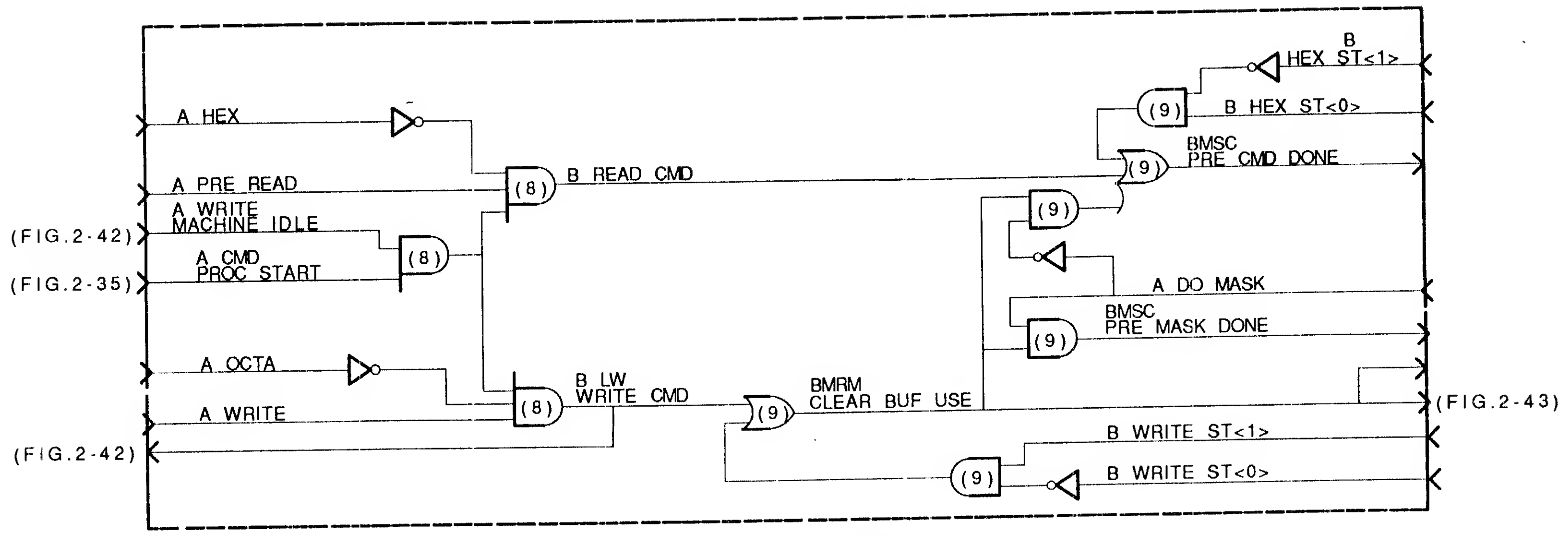
A hex state of 0:1 results in the assertion of BMSC PRE CMD DONE.

C. Normal Write Commands Done

The completion of MSC processing of a normal or masked longword write command (A WRITE true and A OCTA false) is indicated by the assertion of B LW WRITE CMD. The assertion of B LW WRITE CMD requires that the command data be in the MSC (A CMD PROC START true) and that the write machine be in its idle state (A WRITE MACHINE IDLE true).

---

* The MSC has only to transmit an octaword read command to the array board. The array board then reads the four longwords.



NOTE:
NUMBER DESIGNATIONS IN PARENTHESES REFER TO ENGINEERING DRAWINGS CONTAINING CORRESPONDING LOGIC.

SCI D-353

Figure 2-44 Command Done Logic

The completion of MSC processing of an octaword write command (normal or masked) is indicated by a write state of 1:0 (B WRITE ST<1:0> = 1:0) from the write machine in the write-command logic. A write state of 1:0 indicates the last longword write of an octaword write has started (see Table 2-11).

An asserted B LW WRITE CMD or a 1:0 write state asserts BMRM CLEAR BUF USE signifying the completion of MSC processing of any write command. BMRM CLR BUF USE is sent to the MDBC where it clears the MDB write control logic.

BMRM CLEAR BUF USE is ANDed with a negated A DO MASK (normal write) causing BMSC PRE CMD DONE to assert signifying the completion of MSC processing of any normal write command.

2.10.9.2 BMSC PRE MASK DONE -- BMRM CLR BUF USE is ANDed with an asserted A DO MASK (masked write) causing BMSC PRE MASK DONE to assert. The assertion of BMSC PRE MASK DONE signifies the completion of MSC processing of any masked write command.

2.10.10 Command Parity

Figure 2-34 depicts a command parity generator which generates a command parity bit (BMSC CMD PAR) for the MASC. The MASC uses the command parity bit to generate the final command/address parity bit for the array board.

Parity bit BMSC CMD PAR is generated on the following five MSC command signals:

- A WRITE from the command/address/size buffer.
- AMRM STADDR<1:0> from the starting address logic.
- A LEN from the size logic.
- A HEX from the size logic.
- A HEX ST from the size logic.

2.11 MEMORY SEQUENCE CONTROL 1 (MSC1) MCA

The MSC1 MCA (along with the MSC MCA) recives and processes commands for the array modules. The MSC1 is divided into four functional areas and a block diagram is provided for each area. The four diagrams illustrate the entire MCA. All the MCA input/output signals are shown on the diagrams with each signal having a source or destination referenced, by figure number, to other functional area(s).

The four functional areas are defined below.

- Mask Store -- stores four-bit masks associated with the data longwords in the X and Y portions of the MDB data buffer. Generates the write-enable signals for the MDB W and C write ports.

- Select-Out Buffer Control -- if a target array board is busy, the select-out buffer control will queue up to three select-out signals in a select-out buffer in the MDB address I/O select logic. When the array board is free, the control allows the select signals to pass through the buffer.

- Read Buffer Control -- if a target array board is busy, the read buffer control will queue up to three select-out signals associated with read commands, in a read buffer in the MDB address I/O select logic. In addition, up to three second-octa flags can be queued in a hex read buffer. When the array board is free, the control allows the select bits and the second-octa flags to pass through the buffers.

- MDB Address I/O Select Logic -- generates the address-in select bits that locate the command address in the address buffer portion of the MDB. Also generates the address-out select bits that select the address in the address buffer that is output from the MDB to the array board.

2.11.1 Mask Store (Figure 2-45)

The mask fields associated with each data longword (ADFA MASK<3:0>) are received from the ARID and applied to four Y latches and four X latches.

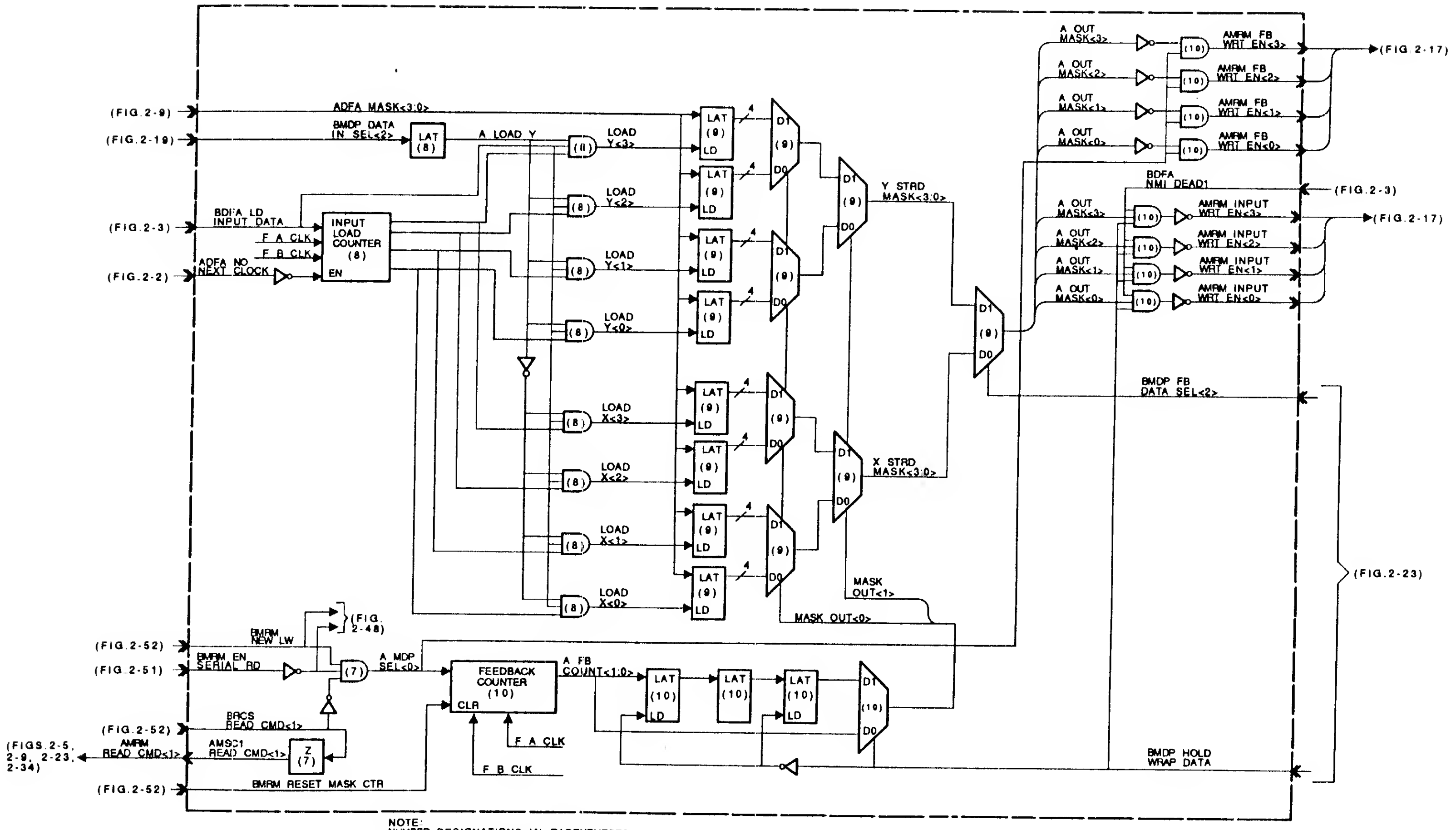
BMDP DATA IN SEL<2> from the MDBC specifies which portion (X or Y) of the MDB data buffer is being loaded with the write data. With BMDP DATA IN SEL<2> asserted (Y buffer being loaded), A LOAD Y asserts to allow the four Y latches to be loaded. When A LOAD Y is false, the four X latches are allowed to load.

BDFA LD INPUT DATA from the FUNK is asserted while the MDB is being loaded. BDFA LD INPUT DATA allows all the latches to be loaded, and also starts an input-load counter. Once started, the counter is incremented each bus cycle by F A CLK and F B CLK to sequence through four outputs which enable the load inputs to the latches. When the Y latches are enabled and an octaword-write operation is being executed, LOAD Y<0> through LOAD Y<3> assert, in sequence, to load the Y latches. For a longword-write operation, BDFA LD INPUT DATA is asserted for only one cycle, hence only LOAD Y<0> asserts to load the first Y latch.

A similar action occurs when the X latches are enabled.

ADFA NO NEXT CLOCK asserts only for single-step operation, hence the input-load counter is normally enabled.

The four-bit masks are unloaded through a network of muxes controlled by a two-bit, binary, mask-out code (MASK OUT<1:0>). When the code is 0:0, the masks in latch 0 (both X and Y) are



NOTE:
NUMBER DESIGNATIONS IN PARENTHESES REFER TO ENGINEERING DRAWINGS CONTAINING CORRESPONDING LOGIC.

SCLD 356

Figure 2-45 Mask Store Block Diagram

unloaded to become Y STRD MASK<0> and X STRD MASK<0>. If an octaword write is executing, the code is incremented in binary format to unload the other latches in sequence.

The MASK OUT code is obtained from a feedback counter started by A MDP SEL<0> when read data associated with a masked write operation is obtained. BRCS READ CMD<1> is false when read data is associated with a masked write operation (Table 2-6). Once started, the counter output is incremented each bus cycle, in binary format, by F A CLK and F B CLK.

BMDP FB DATA SEL<2> from the MDBC asserts when the Y buffer in the MDB is selected for a data unload. Accordingly, Y STRD MASK<3:0> is selected as A OUT MASK<3:0>. When the X buffer is selected for a data unload, X STRD MASK<3:0> is selected as A OUT MASK<3:0>.

A OUT MASK<3:0> drives the write enable signals for both the W and C write ports of the MDB. AMRM FB WRT EN<3:0> enables bytes of read data from the array board to overwrite longword bytes of masked-write data in the MDB via the C write port (Figure 2-17). An AMRM FB WRT EN<3:0> output requires that the read data be associated with a masked write operation (A MDP SEL<0> true). An asserted A OUT MASK bit negates the corresponding AMRM FB WRT EN bit. Hence, the associated byte of read data is not written into the MDB and the masked-write byte remains intact. A negated A OUT MASK bit asserts the corresponding AMRM FB WRT EN bit, causing the associated byte of read data to be written into the MDB and thereby returned to the arrays.

AMRM INPUT WRT EN<3:0> enables bytes of data to be written into the MDB via the W write port. For write operations (both normal and masked), BMDP HOLD WRAP DATA is false and all four AMRM INPUT WRT EN bits are asserted to write the entire longword from the NMI (via the DAD) into the MDB.

When the read data associated with a masked-write operation is routed through the DAD for single-bit error correction, it is returned to the MDB and is also loaded in via the W write port. The corrected read data is byte-loaded into the MDB according to the mask bits.

BMDP HOLD WRAP DATA is asserted by the MDBC to hold the corrected read data in the DAD until NMI traffic has stopped. BDFA NMI DEAD1 is asserted by the FUNK when NMI traffic has stopped. The true state of these two signals make the AMRM INPUT WRT EN bits a function of the A OUT MASK bits. An asserted A OUT MASK bit negates the corresponding AMRM INPUT WRT EN bit. Hence, the associated byte of read data is not written into the MDB and the masked-write byte remains intact. A negated A OUT MASK bit asserts the corresponding AMRM INPUT WRT EN bit, causing the associated byte of read data to be written into the MDB and thus returned to the arrays.

For normal writes and masked writes with no single-bit error correction required, BMDP HOLD WRAP DATA is false and the mask-out code (MASK OUT<1:0>) is obtained directly from the feedback counter. If single-bit error correction was required, BMDP HOLD WRAP DATA is asserted and a delayed mask-out code is obtained from three delay latches. The code is delayed and latched to select the proper mask for the corrected read data after it is returned from the DAD.

2.11.2 Select Out Buffer Control (Figure 2-46)

The assertion of BDFA NEW CMD EARLY signifies a new valid address is ready to be written into the MDB. If there is no parity error associated with the new command (ADFA PARITY ERROR2 false), B NEW CMD asserts. B NEW CMD informs the MDB address I/O select logic to output an MDB select signal for the new address.

In addition, B NEW CMD is clocked through two latches to become B VALID CMD<1>. If the array board has accepted the BNUM probe (BMAS CMD ACPT true from the MASC) and the MSC has completed processing the command (BMSC PRE CMD DONE true from the MSC), B CMD DONE asserts. B CMD DONE prevents B VALID CMD<1> from reaching the third buffer latch, hence, A VALID CMD<2> remains false thereby supplying an asserted A FIRST signal to the MDB address I/O select logic. A FIRST bypasses the select signal around the select-out buffer latches in the logic.

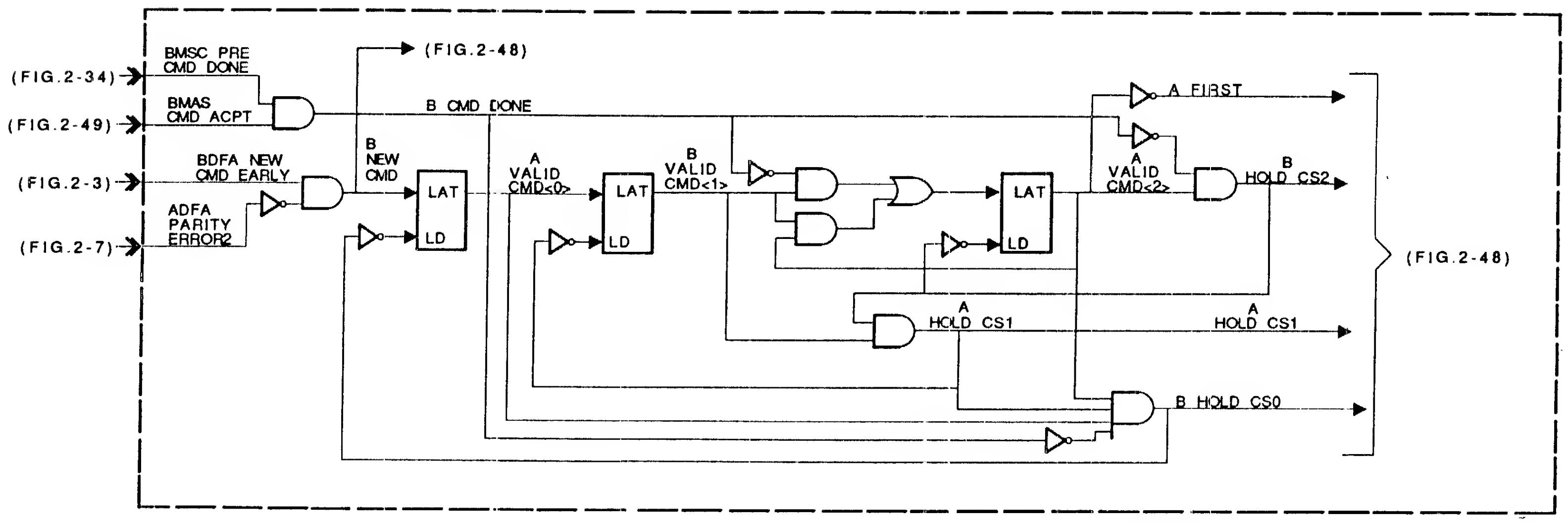
If the array board is not yet able to accept the command or the MSC is still processing the command (as in the case of an octaword write or a hexword read), B CMD DONE will be false. In this case the third buffer latch is set and A VALID CMD<2> asserts. The assertion of A VALID CMD<2> will:

- Negate A FIRST to remove the bypass path in the select-out buffer in the MDB address I/O select logic.
- Assert B HOLD CS2 to the MDB address I/O select logic to latch the select signal in the select-out buffer.
- Latch A VALID CMD<2> to maintain B HOLD CS2 asserted.

B HOLD CS2 is applied to an A-HOLD-CS1 AND gate which looks for another command address as indicated by the assertion of B VALID CMD<1>.

If another command address is received, BDFA NEW CMD EARLY asserts, followed by the assertion of A VALID CMD<0> and B VALID CMD<1>. B VALID CMD<1> enables the A-HOLD-CS1 AND gate asserting A HOLD CS1 to the MDB address I/O select logic. A HOLD CS1:

- Latches the second select signal in the select-out buffer.
- Latches B VALID CMD<1> to maintain A HOLD CS1 asserted.



NOTE:
THE LOGIC IN THIS FIGURE IS CONTAINED ON SHEET 3 OF THE ENGINEERING DRAWINGS.

SCLD-371

Figure 2-46 Select-Out Buffer Control Block Diagram

A VALID CMD<2> and A HOLD CS1 are applied to a B-HOLD-CS0 AND gate which looks for another command as indicated by the assertion of A VALID CMD<0>. If still another command is accepted (see Section 2.10.1.1) before the array board is free, BDFA NEW CMD EARLY and A VALID CMD<0> asserts. The assertion of A VALID CMD<0> causes B HOLD CS0 to assert which:

- Latches the third select signal in the select-out buffer.
- Latches A VALID CMD<0> to maintain B HOLD CS0 asserted.

When the array board is free (BMAS CMD ACPT true) and the MSC has finished processing the current command (BMSC PRE CMD DONE true), B CMD DONE asserts causing the three HOLD-CS latch signals to negate. This allows the select signals to output the select-buffer in the MDB address I/O select logic in order.

2.11.3 Read Buffer Control (Figure 2-47)

When a read command is received (BMRM READ true), and the BNUM pointer and command on the NAB are valid (BMAS BD VALID true), a latch is set asserting A VALID RD<0>. A VALID RD<0> becomes B VALID RD<1> which then becomes A VALID RD<2>. A POP is received from the MDB address I/O select logic and is asserted every time a new read command executes. If A POP is false when A VALID RD<2> asserts, the read command didn't execute (array board busy) and B HOLD2 asserts to the MDB address I/O select logic to latch the MDB select bits associated with the pending read command. B HOLD2 latches A VALID RD<2> to maintain the B HOLD2 latch signal.

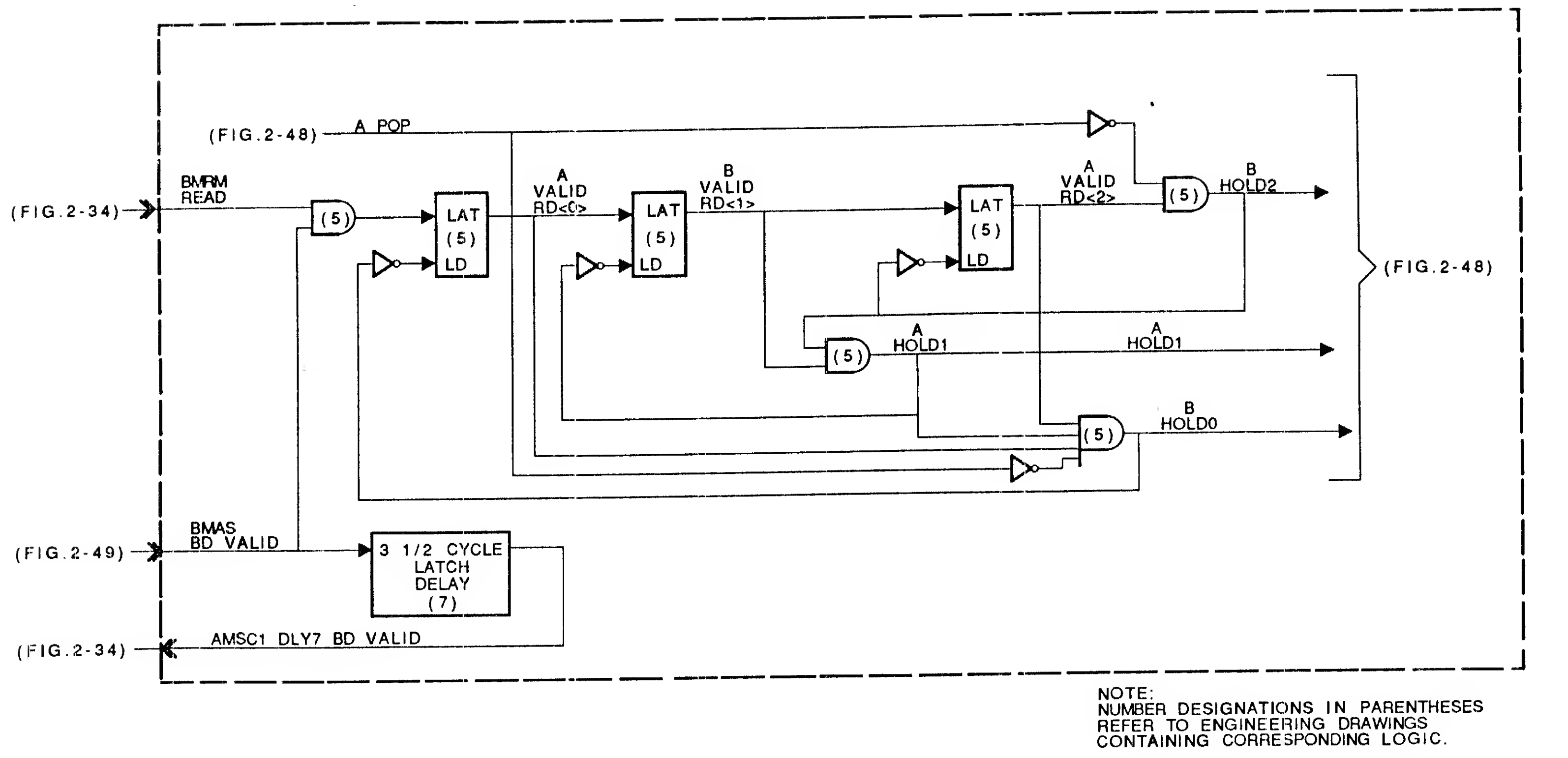
If another read command is received while A POP is still false, A VALID RD<0> asserts causing B VALID RD<1> to assert. B VALID RD<1> causes A HOLD1 to assert and latch the select signal associated with the second read command. A HOLD1 latches up B VALID RD<1> to maintain the A HOLD1 latch signal.

If a third read command is accepted, A VALID RD<0> and B HOLD0 are asserted to latch up the third select signal.

When A POP does assert, B HOLD2 negates, in turn negating A HOLD1 and B HOLD0, allowing the MDB select bits to proceed through the read buffer, and the second-octa flag(s) to pass through the hex-read buffer.

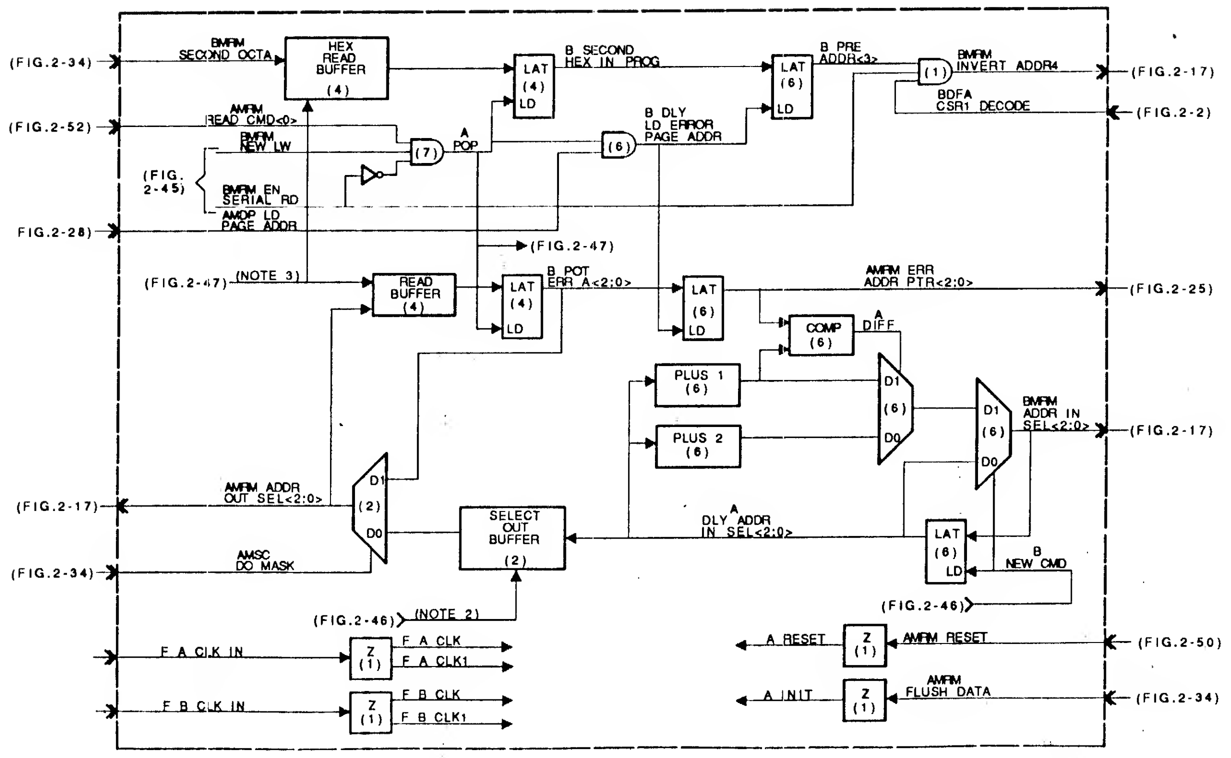
2.11.4 MDB Address I/O Select Logic (Figure 2-48)

2.11.4.1 MDB Address In Select Bits -- The 3-bit MDB address-in select signal (BMRM ADDR IN SEL<2:0>) increments through the eight locations of the MDB address buffer placing the command address in suceeding locations in the buffer. Command addresses are unloaded from the MDB right after being loaded, hence, succeeding locations are always available except for one location reserved for the error page address. If the logic detects that the next location



SCLD-373

Figure 2-47 Read Buffer Control Block Diagram



NOTES:

1. NUMBER DESIGNATIONS IN PARENTHESES REFER TO ENGINEERING DRAWINGS CONTAINING CORRESPONDING LOGIC.
2. A FIRST
   B HOLD CS2
   A HOLD CS1
   B HOLD CS0
3. B HOLD2
   A HOLD1
   B HOLD0

SCLD 367

Figure 2-48 MDB Address I/O Select Block Diagram

contains the error page address, it increments the select bits by two thereby skipping over the location of the error page address.

B NEW CMD asserts when a valid new command is received. The assertion of B NEW CMD will:

- Cause the BMRM ADDR IN SEL<2:0> output mux to select the D1 input for the BMRM ADDR IN SEL<2:0> output.
- Loads BMRM ADDR IN SEL<2:0> into a latch where it becomes A DLY ADDR IN SEL<2:0>.

A DLY ADDR IN SEL<2:0> is incremented by 1 in plus-1 logic, and by 2 in plus-2 logic. The plus-1 incremented select bits are made available at the D1 input of a DIFF mux. They are also applied to a comparator where they are compared to the error address pointer that specifies the buffer location of the address containing a bit error. If the two sets of select bits are not equal, the comparator asserts A DIFF which selects the plus-1 incremented select bits as the BMRM ADDR IN SEL<2:0> output for the next B NEW CMD. If they are equal, A DIFF is false and the plus-2 incremented select bits are selected as the BMRM ADDR IN SEL<2:0> output for the next new command.

When B NEW CMD negates, the BMRM ADDR IN SEL<2:0> select bits are maintained by feedback of A DLY ADDR IN SEL<2:0> through the D0 input of the output mux.

2.11.4.2 **MDB Address Out Select Bits** -- A DLY ADDR IN SEL<2:0> is applied to a three-stage, select-out buffer which can queue up to three sets of select bits if the array boards are not ready to accept commands from the MCL. B HOLD CS2, A HOLD CS1, and B HOLD CS0 are received from the select-out buffer control which asserts them, in order, if the array board(s) are busy. The three hold-signals latch the select signals associated with the commands that are queued. If no signals are queued, the select-out buffer control asserts A FIRST which bypasses the select signal around the buffer and applies it directly to the AMRM ADDR OUT SEL<2:0> output mux where it is selected as the AMRM ADDR OUT SEL<2:0> output.

AMRM ADDR OUT SEL<2:0> is applied to a three-stage read buffer where up to three select signals associated with read commands, can be latched. The latching signals (B HOLD2, A HOLD1, and B HOLD0) are asserted by the read-buffer control when read commands are sensed but the target array board(s) are busy. The output from the read buffer is loaded into a latch by A POP when a longword of read data is received from the arrays (BMRM NEW LW true and BMRM EN SERIAL RD false) and the longword is the first longword of the transfer (AMRM READ CMD<0> true). The latch output (B POT ERR A<2:0>) is a stored set of select bits that represent a potential error address (if a bit error is detected in the read data), and, if this is a masked write operation, serves as the MDB select bits for the write portion of the masked write.

If this is a masked write operation, AMSC DO MASK is asserted by the MSC when the write portion initiates. AMSC DO MASK selects the latched B POT ERR A<2:0> for the AMRM ADDR OUT SEL<2:0> output. B POT ERR A<2:0> were the select-out bits for the read portion of the masked write. Now they are being used as the select-out bits for the write portion.

2.11.5 Error Address Pointer

When the DCHK detects a bit error in the read data, it asserts AMDP LD PAGE ADDR. AMDP LD PAGE ADDR ANDs with A POP to assert A DLY LD ERROR PAGE ADDR when the read data is found to contain a bit error. B DLY LD ERROR PAGE ADDR loads the select bits associated with the error (B POT ERR A<2:0>) into a latch which outputs the error select bits as an error pointer (AMRM ERR ADDR PTR<2:0>).

AMRM ERR ADDR PTR<2:0> is sent to the MDBC as the location in the MDB containing the address of the erroneous data. AMRM ERR ADDR PTR<2:0> is also applied to a comparator to reserve the pointer location in the MDB as discussed in Section 2.11.4.1.

2.11.6 BMRM INVERT ADDR4

BMRM SECOND OCTA is received from the MSC when the second octaword read of a hexword read is executing. BMRM SECOND OCTA is applied to a three-stage, hex-read buffer identical to the three-stage read buffer. Up to three BMRM SECOND OCTA flags can be queued in the buffer by the same three latch signals (B HOLD2, A HOLD1, and B HOLD0) that queued up to three sets of select bits in the read buffer. A POP loads the the output of the hex-read buffer into a latch when the first longword of the second octaword is received. The latch output (B SECOND HEX IN PROG) is loaded into another latch if a bit error is detected during the second octaword read. This causes B PRE ADDR<3> to assert for the erroneous longword.

When CSR1 is read (BDFA CSR1 DECODE and BMRM EN SERIAL RD both true) as a result of the bit error, BMRM INVERT ADDR4 asserts to the MDP where it flips the 4th bit of the error address read out of the MDB. This changes the address to the location of the second octaword where the bit error occurred.

2.12 MEMORY ARRAY SEQUENCE CONTROL (MASC) MCA (Figure 2-49)

2.12.1 Command/Address Parity

The command parity generated in the MSC (BMSC CMD PAR) and the stored address parity from the MDB (BMDP STRD APAR<3:0>) are applied to a parity generator in the MASC. The generator developes a composite parity bit (BMRM CMD ADDR PARITY) over the entire

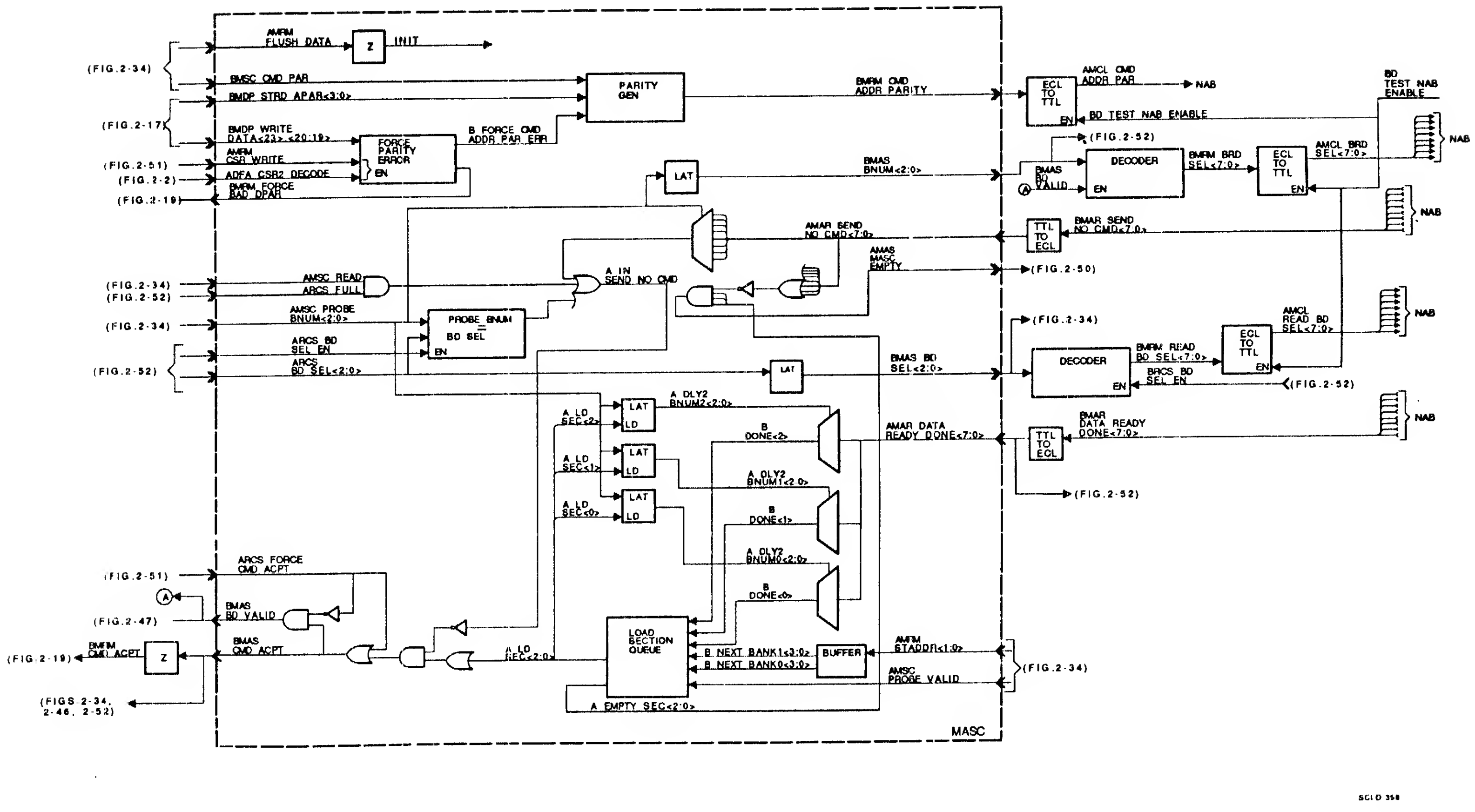


Figure 2-49 MASC Block Diagram

command address field. BMRM CMD ADDR PARITY is converted to TTL and then placed on the NAB as AMCL CMD ADDR PAR.

2.12.2 Force Parity Error

Write data bits BMDP WRITE DATA<23> and BMDP WRITE DATA<20:19> are write only bits of CSR2 and are used to force parity errors for testing. AMRM CSR WRITE and ADFA CSR2 DECODE specify a write of CSR2 by enabling the force-parity-error logic. The assertion of write data bits <23> and <20> assert the forced error function while bit <19> specifies a data forced error or a command/address forced error. When bit <19> is asserted, BMRM FORCE BAD DPAR is asserted to the MDBC where it forces a data parity error. When bit <19> is negated, B FORCE CMD ADDR PAR ERR is asserted to the MASC parity generator where it forces a parity error in the command/address field sent to the array boards.

2.12.3 Board Number (BMAS BNUM<2:0>)

The BNUM probe from the MSC (AMSC PROBE BNUM<2:0>) outputs the MASC as BMAS BNUM<2:0> to the RCS. BMAS BNUM<2:0> is also applied to a decoder which, when enabled by BMAS BD VALID, asserts one of seven BMRM BRD SEL<7:0> lines to the array boards. The asserted BMRM BRD SEL line is converted from ECL to TTL to become AMCL BRD SEL<X>. AMCL BRD SEL<X> enables the selected array board for the commanded operation.

2.12.4 Send No Command

Eight send-no-command lines (AMAR SEND NO CMD<7:0>) are received from the array boards (one line from each board), and are asserted when the respective board cannot accept a command from the MCL. This will occur when the array board is doing a read or a refresh. The send-no-command lines are applied to a mux controlled by the probe board number (AMSC PROBE BNUM<2:0>). The probe board number selects the send-no-command line associated with the probe command. If the line is asserted, A IN SEND NO CMD is asserted (via an OR gate) within the MASC to inhibit command data from being sent to the array board.

A IN SEND NO CMD is also asserted during a read command (AMSC READ true) if the RCS buffers are full (ARCS FULL true).

A third condition that asserts A IN SEND NO CMD is the command probe selecting an array board that is involved in a masked write. If the array board has already been read for the masked write, ARCS BD SEL EN is true and enables a comparator which compares AMSC PROBE BNUM <2:0> with ARCS BD SEL<2:0>. ARCS BD SEL<2:0> is the board number being saved by the RCS for the write portion of the masked write. If the two are equal, A IN SEND NO CMD asserts to inhibit the transfer of the new command data to the array board.

2.12.5 MASC Empty

If all the array boards are free, as indicated by all eight AMAR SEND NO CMD<7:0> lines being false, and the MASC queue is empty (A EMPTY SEC<2:0> all true), AMAS MASC EMPTY is asserted to the RCS to inform it of the empty status.

2.12.6 Board Select (BMAS BD SEL<2:0>)

ARCS BD SEL<2:0> from the RCS, specifies an array board containing read data that is is available to the MCL. ARCS BD SEL<2:0> is also used to specify the array board for the write portion of a masked write operation.

During a read operation, the board select code outputs the MASC as BMAS BD SEL<2:0> where it is applied to a decoder. The decoder is enabled by BRCS BD SEL EN from the RCS, and functions to assert one of eight read-board-select lines (AMCL READ BD SEL<7:0>) to the array boards. The asserted AMCL READ BD SEL<7:0> line enables the selected array board to send its read data to the MCL.

During a masked write operation, the decoder is not enabled. In this case, BMAS BD SEL<2:0> is used by the MSC as the board number for the write portion of the masked write.

2.12.7 Command Accept (BMAS CMD ACPT) and Board Valid (BMAS BD VALID)

BMAS CMD ACPT signifies to the MCL that the probe command was accepted by the array board and processing of the command can proceed. BMAS BD VALID signifies that the BNUM probe and command data on the NAB are valid.

For CSR operations, the RCS asserts ARCS FORCE CMD ACPT which asserts BMAS CMD ACPT but negates BMAS BD VALID. The assertion of BMAS CMD ACPT initiates the CSR command processing in the MCL. The false state of BMAS BD VALID signifies the NAB data as invalid as the command access is not to the array boards.

For non-CSR operations, the assertion of BMAS CMD ACPT causes BMAS BD VALID to assert.

BMAS CMD ACPT is asserted from the load-section queue by any of the load-section signals (A LD SEC<2:0>), provided A IN SEND NO CMD is false. A IN SEND NO CMD inhibits the processing of a command due to the reasons discussed in Section 2.12.4.

AMSC PROBE VALID is received from the MSC when a new command is received for the array boards. With no commands in the queue (A LD SEC<2:0> all negated), AMSC PROBE VALID causes the load-section queue to assert A LD SEC<2>. A LD SEC<2> loads the AMSC PROBE BNUM<2:0> command probe into a latch. The latch output (A DLY2 BNUM2<2:0>) causes a mux to monitor the data-ready-done line from the array board selected by the BNUM probe. When AMAR DATA RDY

DONE<X> asserts, B DONE<2> is asserted to the load-section queue causing A LD SEC<2> to negate.

The load-section queue can queue up to three commands while waiting for a "done" indication from the array boards. If a second command is received before data-ready-done asserts from the first command, AMSC PROBE VALID re-asserts. In addition, the starting address associated with the command (AMRM STADDR<1:0>) asserts B NEXT BANK1<3:0> according to the array bank selected as the first bank. The asserted next-bank code asserts A LD SEC<1> to load the new BNUM probe into another latch. The latch output causes another mux to monitor the data-ready-done line from the new array board selected by the BNUM probe.

If still another command is accepted before the data-ready-done line is asserted, a similar sequence asserts A LD SEC<0> to load the third BNUM probe into a third latch. The latch output contols a third mux to monitor the data-ready-done line from the third array board.

B DONE<2:0> from the muxes, assert in sequence when their respective data-ready-done lines are asserted.

A EMPTY SEC<2:0> indicates the full/empty status of the load-section queue.

2.13 READ CONTROL SEQUENCER (RCS) MCA

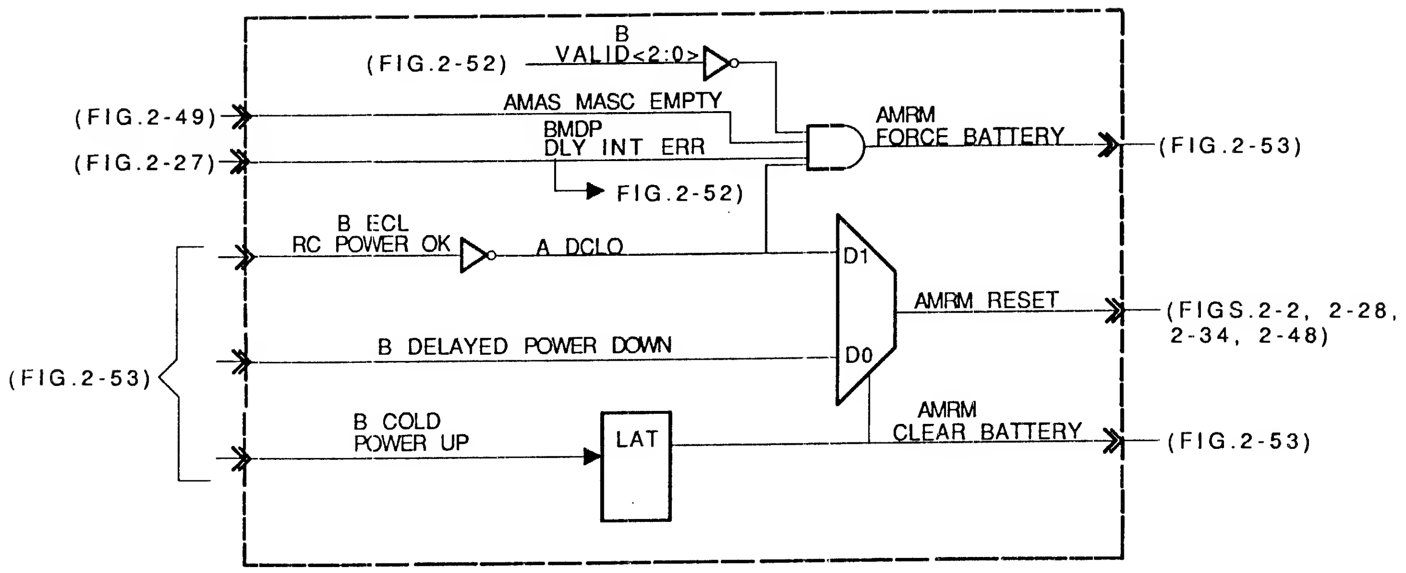
Three block diagrams (Figures 2-50, 2-51, and 2-52) cover the RCS MCA. All the RCS inputs and outputs appear on the three diagrams. These inputs and outputs are referenced to other figures of the memory system description.

2.13.1 Power Control (Figure 2-50)

When system power is interrupted, B ECL RC POWER OK from the battery backup unit (BBU) is negated causing the assertion of A DCLO in the RCS. A DCLO is applied to an AND gate where it checks that:

- All in-process writes have been completed (AMAS MASC EMPTY true).
- All reads have been aborted (RCS buffer is empty as indicated by B VALID<2:0> all false).
- An internal error signal has been asserted to the MCL (BMDP DLY INT ERR true).

It then asserts AMRM FORCE BATTERY to the BBU which proceeds to place the array boards into battery mode of operation. The BBU then asserts B DELAYED POWER DOWN back to the RCS. B DELAYED POWER DOWN asserts AMRM RESET via a reset mux. AMRM RESET is distributed throughout the MCL to reset logic and clear queues.



SCLD-352

Figure 2-50 Power Control

When power returns, B COLD POWER UP from the BBU indicates if the array data was maintained by battery power during the power outage, or if the duration of the power outage was too long and the array data was lost (powering-up from this state is a cold power up). If a cold power up is executing (B COLD POWER UP true), AMRM CLEAR BATTERY is asserted to the BBU where it prepares the BBU to return to normal operation. AMRM CLEAR BATTERY also switches the reset mux so that AMRM RESET becomes a function of B ECL RC POWER OK. The BBU keeps B ECL RC POWER OK negated during the cold power up thereby holding AMRM RESET asserted. When the cold power up has completed, B ECL RC POWER OK asserts, negating AMRM RESET.

If the power up is from battery mode (B COLD POWER UP FALSE), AMRM RESET is a function of B DELAYED POWER DOWN. B DELAYED POWER DOWN is immediately negated when power returns, causing AMRM RESET to negate via the reset mux.

2.13.2 CSR Control (Figure 2-51)

A CSR state machine generates a two-bit state code (A CSR ST<1:0>) which controls the CSR signals generated by the RCS.

The state machine idles in state 0 (A CSR ST<1:0> = 0:0). When A CSR PROBE VALID asserts, the machine is enabled to move into other states. Factors that determine the state of the machine are:

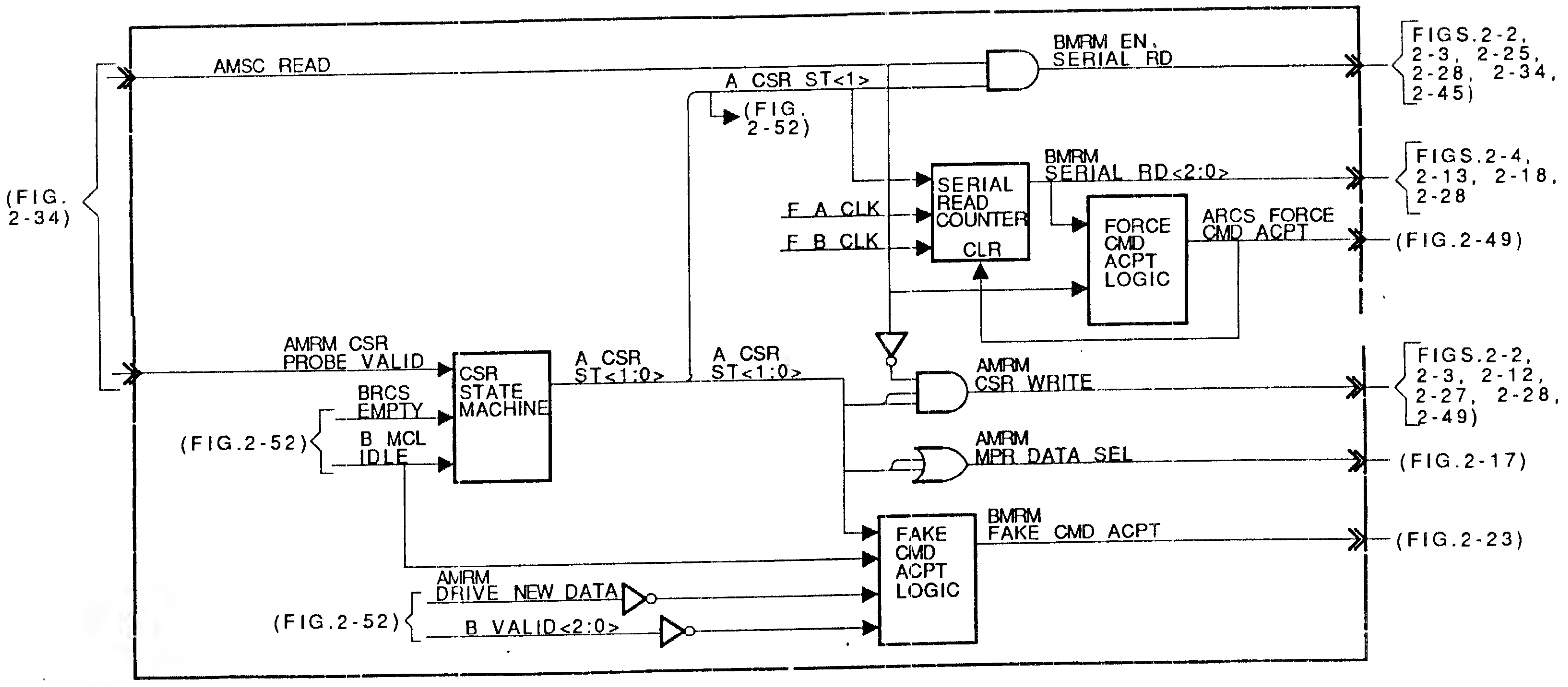
- The full/empty state of the RCS buffer (BRCS EMPTY).
- The idle state of the MCL (B MCL IDLE).

2.13.2.1 BMRM EN SERIAL RD -- BMRM EN SERIAL RD enables CSR bits to be output from various locations throughout the MCL. It starts CSR internal read counters to output the CSR data in a serial format.

BMRM EN SERIAL RD is asserted during a read command (AMSC READ true) when the CSR state machine is in state 2 or 3.

2.13.2.2 BMRM SERIAL RD<2:0> -- BMRM SERIAL RD<2:0> specifies to the MCL what CSR bits are to be read out during a CSR read. BMRM SERIAL RD<2:0> is obtained from a serial read counter which is enabled in CSR state 2 or 3 (while BMRM EN SERIAL RD is asserted). When enabled, the counter output is incremented by F A CLK and F B CLK. The counter is cleared by ARCS FORCE CMD ACPT from the force-command-accept logic.

2.13.2.3 AMRM CSR WRITE -- AMRM CSR WRITE is asserted to inform the MCL to write CSR data. It is asserted in CSR state 3 during a write command (AMSC READ false).



SCLD-349

Figure 2-51 CSR Control

2.13.2.4 ARCS FORCE CMD ACPT -- A RCS FORCE CMD ACPT is asserted to the MASC during CSR operations to force the assertion of BMAS CMD ACPT. For CSR reads, ARCS FORCE CMD ACPT asserts every eight read bits. For CSR writes, ARCS FORCE CMD ACT asserts on the fourth cycle (third count of BMRM SERIAL RD<2:0>) and then resets the counter.

2.13.2.5 AMRM MPR DATA SEL -- AMRM MPR DATA SEL is asserted for all CSR operations (CSR states 1, 2, and 3). It is sent to the MDB where it routes the MDB data to the DFA.

2.13.2.6 BMRM FAKE CMD ACPT -- BMRM FAKE CMD ACPT is sent to the MDBC where it asserts the MDB strobe to enable the CSR data from the MDB to the DFA. It is asserted in CSR state 1 when the MCL is idle (B MCL IDLE true), and read data is not being taken from the array boards (AMRM DRIVE NEW DATA false), and there is no command data queued in the RCS queue buffer (B VALID<2:0> all false).

2.13.3 Read Command Bits (Figure 2-52)

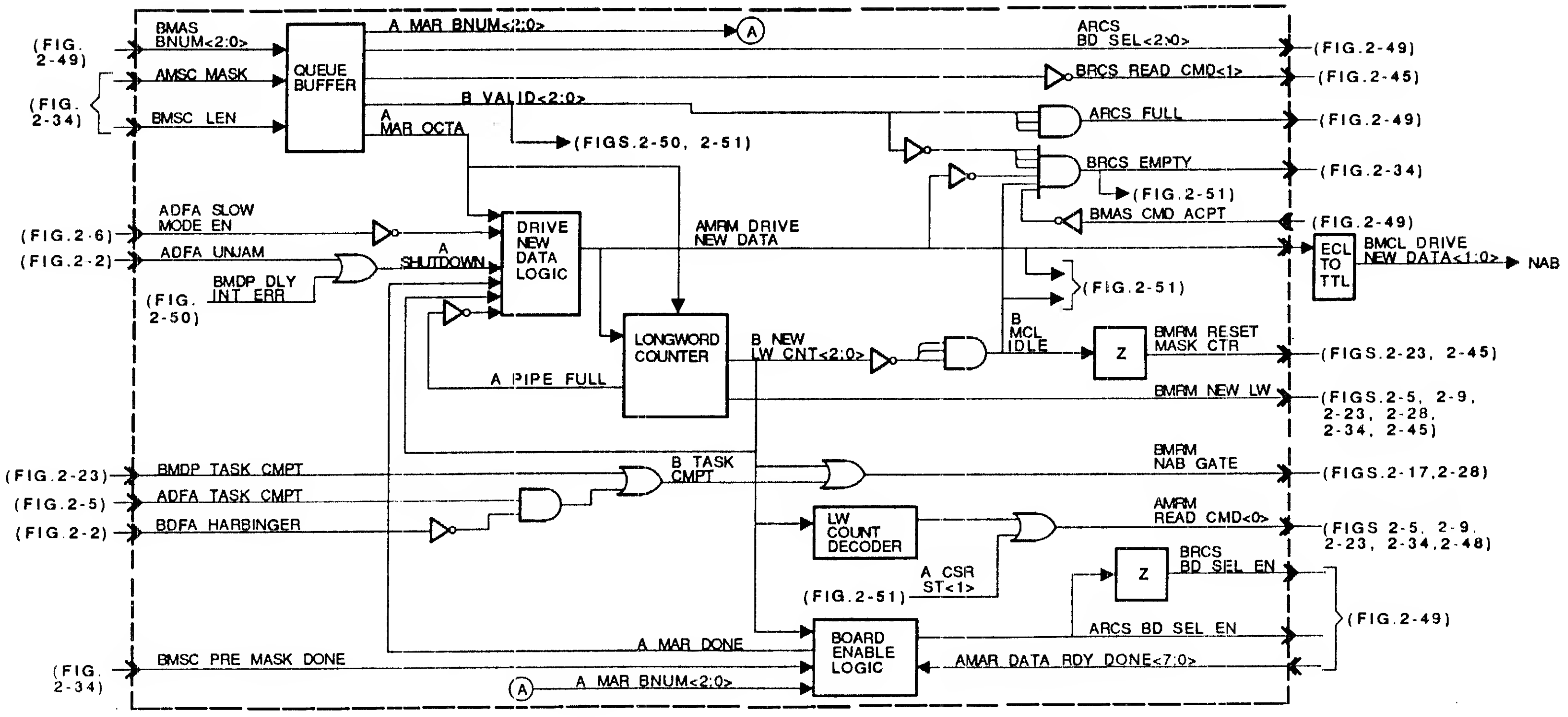
2.13.3.1 AMRM READ CMD<0> -- AMRM READ CMD<0> specifies a longword of read data as being the first longword of a command, or a "next" longword. The cases of CSR reads and normal reads are considered.

- For a CSR read, AMRM READ CMD<0> is asserted in CSR states 2 or 3 (A CSR ST<1> true) when the CSR longword is read.
- For a normal read, the output of the longword count decoder is examined. The decoder decodes the longword count and asserts its output (causing AMRM READ CMD<0> to assert) for longword 0. The decoder output is negated for longwords 1, 2, and 3.

2.13.3.2 BRCS READ CMD<1> -- BRCS READ CMD<1> specifies a longword of read data as being for the MDP (masked) or for the DFA (normal). BRCS READ CMD<1> is the AMSC MASK signal from the MSC, that has been routed through the three-stage, queue-buffer and then inverted. Consequently, read bit <1> is asserted for normal reads (AMSC MASK false) and negated for masked reads (AMSC MASK true).

2.13.4 Board Select/Enable (Figure 2-52)

2.13.4.1 Board Select -- ARCS BD SEL<2:0> is the command probe number received from the MASC as BMAS BNUM<2:0> and routed through the queue buffer. It is returned to the MASC as the board select number.



SCLD-345 →

Figure 2-52 Array Read Control

2.13.4.2 Board Select Enable -- Board enable logic generates two board-select enable signals which validate the board-select signal for the array board. ARCS BD SEL EN enables a comparator in the MASC where the number of the board being read (ARCS BD SEL<2:0>) is compared with the probe number of the board to be accessed for a new command.

BRCS BD SEL EN enables a decoder which converts the current three-bit board select code from the MASC into an enabled board-select line used to enable the selected array board.

Three signal areas are monitored by the board-enable logic to determine the period to hold a board enabled. These are:

- The longword count from the longword counter -- to see if more longwords are yet to come from the array board.
- AMAR DATA RDY DONE<7:0> from the NAB -- to see if the array board has completed the current operation.
- BMSC PRE MASK DONE from the MSC -- to see if the array board should be held enabled for the write portion of a masked write operation.

2.13.5 Read Data In Signals (Figure 2-52)

2.13.5.1 AMRM DRIVE NEW DATA -- AMRM DRIVE NEW DATA is asserted to the array board instructing it to place a longword of read data on the NAB. To determine if the array board has read data to be taken, the drive-new-data logic monitors the longword count (B NEW LW CNT<2:0>), and if this is an octaword transfer (A MAR OCTA). If these sugnals indicate that more read data is to come from the array board, and that the MCL can accept new read data (A PIPE FULL false), it asserts AMRM DRIVE NEW DATA.

When a multiword read is executing, a longword counter is enabled and outputs a three-bit count (B NEW LW CNT<2:0>) set for the number of longwords to be transferred. AMRM DRIVE NEW DATA decrements the counter each time a longword of read data is requested from the array board. When decremented, the counter asserts BMRM NEW LW informing the MCL that a new longword of read data is coming. The output count (B NEW LW CNT<2:0>) provides an indication of the state of the read transfer by specifying the number of longwords yet to come.

When the longword counter reaches 0, B MCL IDLE asserts which in turn asserts BMRM RESET MASK CTR to the MSC1. BMRM RESET MASK CTR clears the feedback counter in the MSC1 mask-store logic.

2.13.5.2 BMRM NAB GATE -- BMRM NAB GATE enables read data from the NAB to pass to the DFA for a normal read, and to the MDB for the read portion of a masked write. BMRM NAB GATE is asserted if there is a count left in the longword counter, or B TASK CMPT is true (preceding read data processed and MCL ready for more).

2.13.6 RCS Full/Empty Status (Figure 2-52)

2.13.6.1 ARCS FULL -- Command data queued in the RCS buffer is monitored by looking at the B VALID<2:0> latch bits. If all three are asserted, the buffer is full and ARCS FULL is asserted to the MASC where it causes the MASC send-no-command signal to assert.

2.13.6.2 BRCS EMPTY -- BRCS EMPTY is asserted to the MSC to allow the memory system to return from an internal-error state. The assertion of BRCS EMPTY requires that:

- The RCS buffer is empty (B VALID<2:0> all false).
- No read data is currently being requested from the array boards (AMRM DRIVE NEW DATA false).
- No longwords of read data are yet to be obtained from the array boards (B MCL IDLE true).
- There are no new read commands that have been accepted by the array boards (BMAS CMD ACPT false).

2.14 BATTERY BACKUP UNIT (BBU)

The battery backup unit performs three functions:

1. Supplies the refresh clock to the array boards for both normal refreshes and battery-mode refreshes. The BBU contains a 67 KHz refresh oscillator that generates the refresh clock. The clock period is 14.9 microseconds.
2. Upon sensing a loss of power, places the memory system into battery mode.
3. During a power up, notifies the memory system whether the power up is from battery mode (battery power maintained array data during power outage), or if it is a cold start (power outage too long and array data is lost*).

Figure 2-53 is a block diagram of the MCL BBU.

Power supplied to the BBU is +5 V from the system BBU +5 V module.

* Battery power will maintain array data for at least ten minutes.

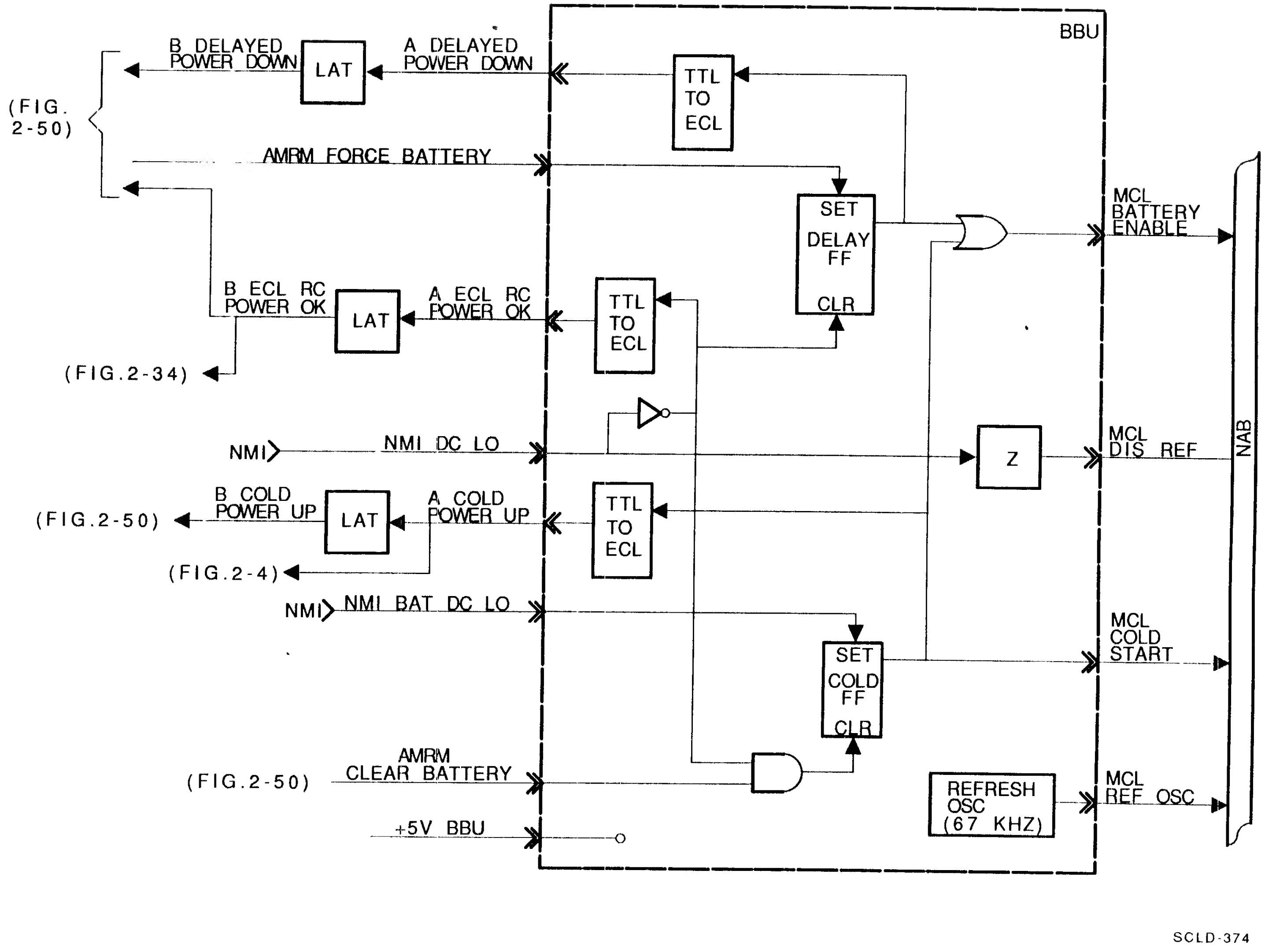


Figure 2-53 MCL BBU Block Diagram

2.14.1 Loss of Power

Figure 2-54 is a flow diagram of the BBU power-down sequence. When system power is interrupted, NMI DC LO is asserted to the BBU. The assertion of NMI DC LO does the following:

- Asserts MCL DIS REF to the refresh logic in the array boards where it disables normal refresh operations.
- Removes the clear input from the BBU delay flip-flop.
- Negates B ECL RC POWER OK to the MSC and the RCS.

When the MSC senses the negation of B ECL RC POWER OK, it initiates the power-down sequence by asserting AMRM FLUSH DATA and AMRM INT ERR.

When the RCS senses the negation of B ECL RC POWER OK, it checks that all in-process writes are completed, all reads have been aborted, and that all the array boards are not busy. It then asserts AMRM FORCE BATTERY to the BBU which sets the delay flip-flop. Setting the delay flip-flop asserts:

- B DELAYED POWER DOWN back to the RCS.
- MCL BATTERY ENABLE to the refresh logic in the array boards where it initiates battery-mode refreshes.

Battery-mode refreshes continue so long as the +5 V BBU battery power stays up. This is guaranteed to be at least ten minutes. If power returns while the BBU +5 V is still operational, a power up from battery mode will occur. If the BBU +5 V is not operational when power returns, a "cold start" power up will execute.

2.14.2 Return of Power

Figure 2-55 is a flow diagram of the BBU power-up sequence. The return of system power is indicated by the negation of NMI DC LO. The power-up sequence varies according to whether the power up is from battery mode or is a cold start.

If the power up is from battery mode, the BBU +5 V from the power system was operational during the power loss and the memory system is operating in battery mode. Here, the negation of NMI DC LO:

- Negates B DELAYED POWER DOWN to the RCS.
- Negates MCL BATTERY ENABLE to take the array boards out of battery mode.
- Negates MCL DIS REF so the array boards may execute normal refreshes.
- Asserts B ECL RC POWER OK to the RCS and MSC informing them that system power is operational.

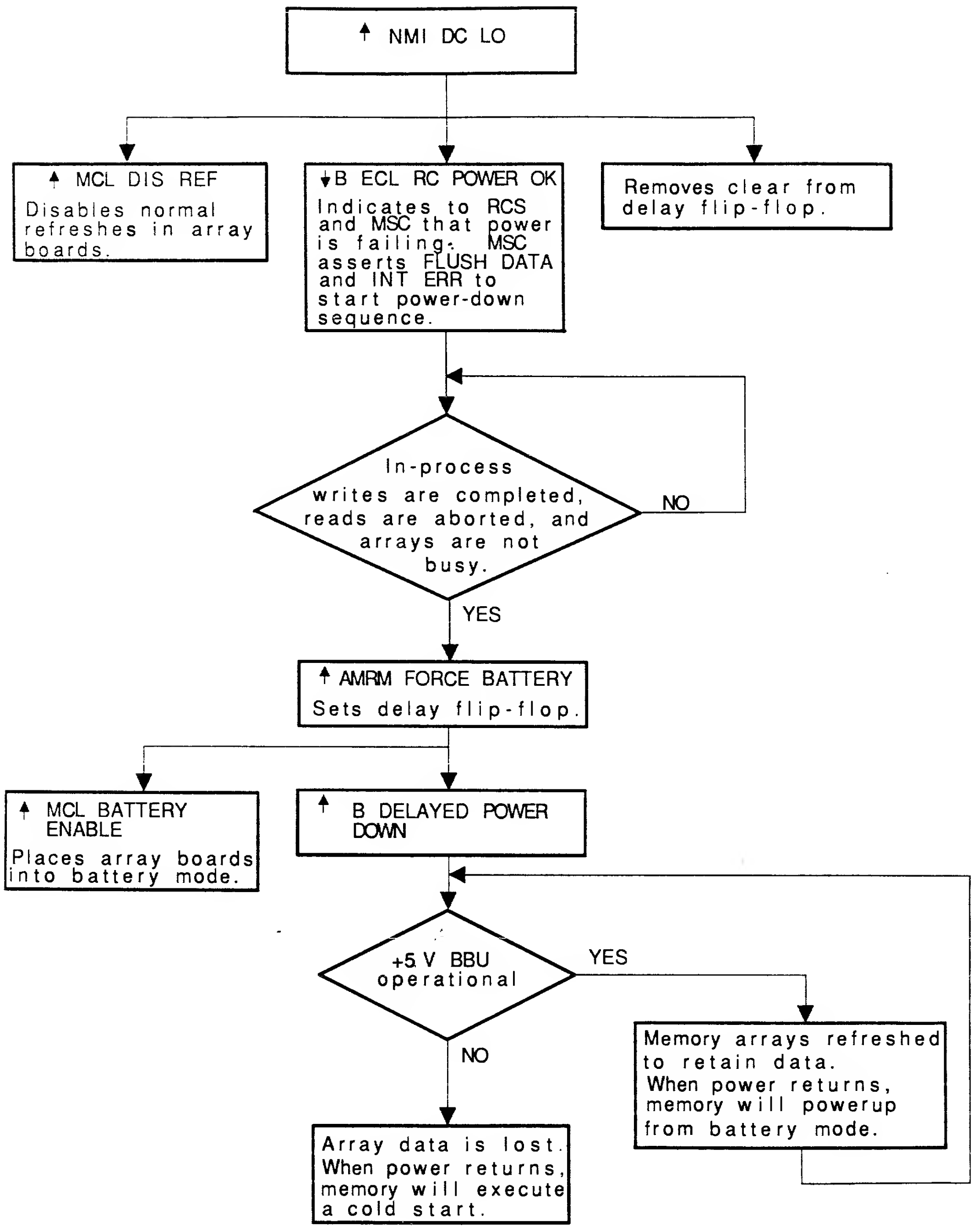

↑ NMI DC LO
↑ MCL DIS REF
Disables normal refreshes in array boards.
↓B ECL RC POWER OK
Indicates to RCS and MSC that power is failing. MSC asserts FLUSH DATA and INT ERR to start power-down sequence.
Removes clear from delay flip-flop.
In-process writes are completed, reads are aborted, and arrays are not busy.
NO
YES
↑ AMRM FORCE BATTERY
Sets delay flip-flop.
↑ MCL BATTERY ENABLE
Places array boards into battery mode.
↑ B DELAYED POWER DOWN
+5 V BBU operational
YES
NO
Memory arrays refreshed to retain data. When power returns, memory will powerup from battery mode.
Array data is lost. When power returns, memory will execute a cold start.
SCLD-375

Figure 2-54 Power Down Flow Diagram

If the power up is a cold start, the BBU +5 V from the power system was not operational during the entire power loss and the array data is lost. After the BBU +5 V becomes operational, NMI DC LO negates which will assert B ECL RC POWER OK and negate B DELAYED POWER DOWN just as in the case of power up from battery

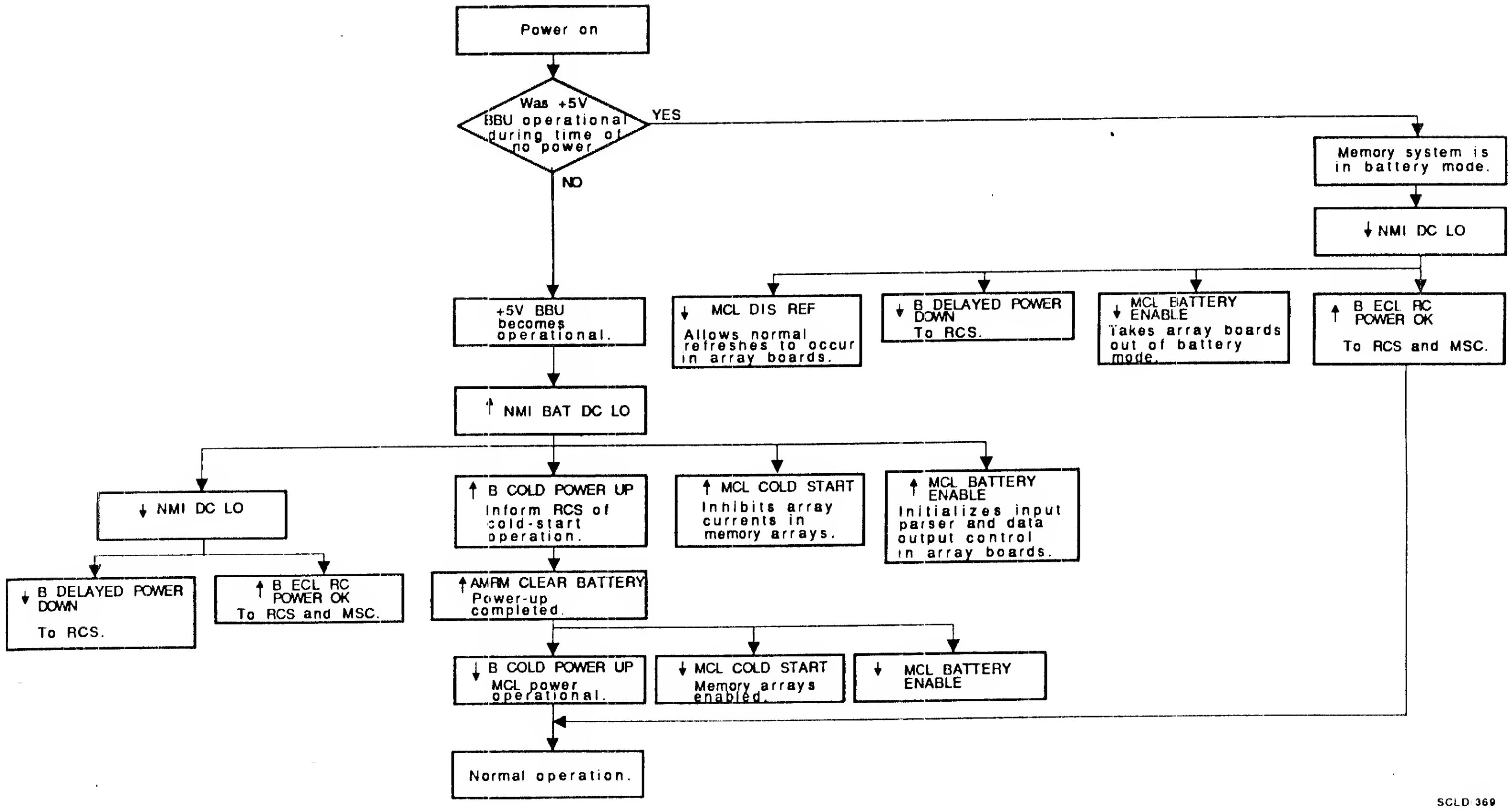

Power on
Was +5V BBU operational during time of no power
YES
NO
Memory system is in battery mode.
↓ NMI DC LO
↓ MCL DIS REF Allows normal refreshes to occur in array boards.
↓ B DELAYED POWER DOWN To RCS.
↓ MCL BATTERY ENABLE Takes array boards out of battery mode.
↑ B ECL RC POWER OK To RCS and MSC.
+5V BBU becomes operational.
↑ NMI BAT DC LO
↓ NMI DC LO
↓ B DELAYED POWER DOWN To RCS.
↑ B ECL RC POWER OK To RCS and MSC.
↑ B COLD POWER UP Inform RCS of cold-start operation.
↑ MCL COLD START Inhibits array currents in memory arrays.
↑ MCL BATTERY ENABLE Initializes input parser and data output control in array boards.
↑ AMRM CLEAR BATTERY Power-up completed.
↓ B COLD POWER UP MCL power operational.
↓ MCL COLD START Memory arrays enabled.
↓ MCL BATTERY ENABLE
Normal operation.
SCLD 369

Figure 2-55 Power Up Flow Diagram

mode. In addition, NMI BAT DC LO asserts signifying this to be a cold start. The assertion of NMI BAT DC LO will:

- Assert MCL COLD START to the array boards to inhibit the flow of array currents. Array data is already lost and blocking array current flow enhances the power-up sequence for the rest of the system.

- Assert MCL BATTERY ENABLE to initialize logic in the input parser and the data-output control in the array boards.

- Assert B COLD POWER UP to the RCS informing it of the cold-start status.

When system power up is completed, the RCS asserts AMRM CLEAR BATTERY to the BBU which negates B COLD POWER UP, MCL COLD START, and MCL BATTERY ENABLE to resume normal operation.

CHAPTER 3
FOUR MEGABYTE MEMORY ARRAY BOARD (MAR4)

3.1 VAX 8800 ARRAY BUS (NAB)

There are 166 signal lines on the NAB bus that carry 21 signals between the MCL and the MAR4 boards. The 21 signals and the number of lines associated with each are listed in Table 3-1. The MCL to MAR4 signals are listed first.

3.1.1 Signal Clocks

Signals and data on the NAB bus are referenced to two clocks designated as phase A and phase B. The clocks have a period of 45 ns and are 180 degrees out of phase.

The MAR4 receives a free-running, phase B clock (MCL ECL F B CLK IN) from the MCL from which it derives all its internal clocks. Consequently signals on the NAB from the MAR4 are referenced to the B clock and are prefixed with the letter B. NAB signals from the MCL are referenced to the A clock and are prefixed with the letter A.

One exception to this is the BMCL DRIVE NEW DATA signal which transfers read data from the MAR4 to the MCL. It is referenced to the B clock so that the read data will appear on the NAB sooner (the next bus cycle; see Section 3.1.3).

Figures 3-1, 3-2, and 3-3 are timing diagrams which show both the phase A and phase B clocks. The NAB signals shown in the diagrams are referenced to the rising edge of the phase A or phase B clock according to their prefix designation.

3.1.2 Longword Write Timing

Figure 3-1 illustrates NAB signal timing for a longword write operation. All the NAB signals used for a write operation are asserted for one bus cycle (45 ns).

In the first bus cycle (command/address cycle), the MCL asserts the board select line for the desired MAR4. It also asserts the command/address and the command/address parity bit. The board select and command/address signals are referenced to the A clock.

Table 3-1 NAB Signals

Signal	Bits	Direction	Function
MCL ECL F B CLK IN	2 *	MCL to MAR4	Supplies phase B clock for MAR4.
AMCL BRD SEL<7:0> +	8	MCL to MAR4	Selects a MAR4 board for a command operation (read or write).
AMCL CMD<3:0>	4	MCL to MAR4	Command
AMCL CMD<3>			Command write bit. Negates to specify a write operation.
AMCL CMD<2>			Command octaword bit. Asserts to specify an octaword read operation.
AMCL CMD<1:0>			Command starting address. Specifies the MAR4 bank to be accessed for a longword read or write. For an octaword read, specifies the first bank to return data to the MCL.
AMCL ADDR<25:4>	22	MCL to MAR4	Specifies location to be accessed within the array(s).
AMCL CMD ADDR PAR	1	MCL to MAR4	Command/address parity bit.
AMCL DATA<31:0>	32	MCL to MAR4	Write data.
AMCL BAD DATA	1	MCL to MAR4	Asserted when write data is bad.
AMCL WRITE ENABLE	1	MCL to MAR4	Asserted when write data is to be written.
AMCL DATA PAR	1	MCL to MAR4	Parity bit for the write data and for the bad data and write enable bits.

* MCL ECL F B CLK IN H and MCL ECL F B CLK IN L.

+ One per MAR4 board.

Table 3-1 NAB Signals (Cont)

Signal	Bits	Direction	Function
AMCL READ BD SEL<7:0> +	8	MCL to MAR4	Selects and prepares a MAR4 board for the transfer of read data to the MCL.
BMCL DRIVE NEW DATA	1	MCL to MAR4	Initiates transfer of read data from a MAR4 bank to the MCL.
MCL REF OSC	©	MCL to MAR4	Refresh clock. Initiates refresh cycles on the MAR4 boards.
MCL BATTERY ENABLE	1	MCL to MAR4	Places MAR4 into battery mode.
MCL DIS REF	1	MCL to MAR4	Inhibits initiation of refresh cycles in battery mode.
MCL COLD START	1	MCL to MAR4	Disables array currents during system power-up.
BMAR CMD ADDR PAR ERR	1	MAR4 to MCL	Command/address parity error.
BMAR DATA RDY DONE<7:0> +	8	MAR4 to MCL	Signifies that the commanded operation has been completed. For a write, indicates the selected array bank has been written. For a read, indicates the selected array bank(s) has (have) been read and data is available in the MAR4.

+ One per MAR4 board.

© MCL REF OSC H to four MAR4s and MCL REF OSC L to other four so all MAR4s will not be refreshing simultaneously.

Table 3-1 NAB Signals (Cont)

Signal	Bits	Direction	Function
BMAR SEND NO CMD<7:0> +	8	MAR4 to MCL	Inhibits commands from MCL.
BMAR DATA<31:0>	32	MAR4 to MCL	Read data (longword).
BMAR DATA<38:32>	7	MAR4 to MCL	Read data (check bits).
MAR MEM SIZE<23:00>	24 @	MAR4 to MCL	Indicates size of memory array board as follows:
			MAR MEM Size <2 1 0>
			0 1 1 = 4 megabytes

+ One per MAR4 board.

@ Three per array board.

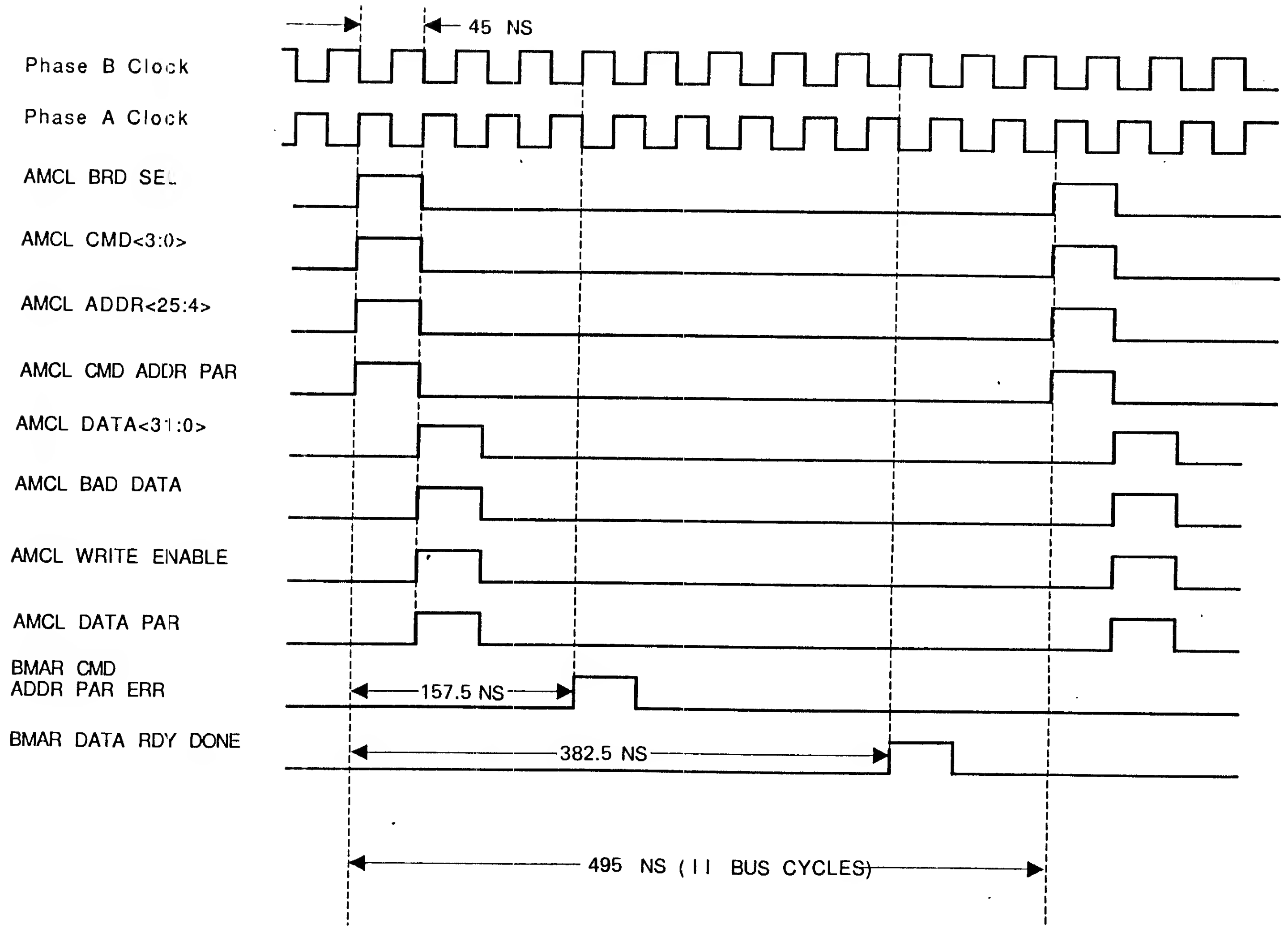
In the second bus cycle (data cycle), the MCL places the data longword, the bad data bit, the write enable bit, and the data parity bit on the NAB.

If the MAR4 detects a parity error in the command/address, it asserts the command/address parity error bit 157.5 ns from time 0. The command/address parity error bit is referenced to the B clock.

Three hundred eighty-two and one-half ns from time 0 the MAR4 asserts the data-ready-done bit indicating it has completed the write operation. If a command/address parity error was detected, the MAR4 will not write the data but the data-ready-done bit will still be asserted.

Sixty-seven and one-half ns after the negation of the data-ready-done bit, the MCL can assert another command to the same bank. As seen in Figure 3-1, the minimum time period between write commands to the same array bank is 495 ns (11 bus cycles).

Consecutive writes to the four banks of a MAR4 board can be made one right after the other. The MCL asserts four sets of board select/command/address/ parity bit signals in four succeeding bus cycles with each bus cycle followed by its data cycle in which the write data, the bad-data bit, the write-enable bit, and the data-parity bit are asserted. Four signal sequences will assert on the NAB with the signals in one sequence displaced one bus cycle from the signals in the following sequence. However, a second command cannot be issued to any one bank until 495 ns after the first command.



SCLD-388

Figure 3-1 Longword Write Timing Diagram

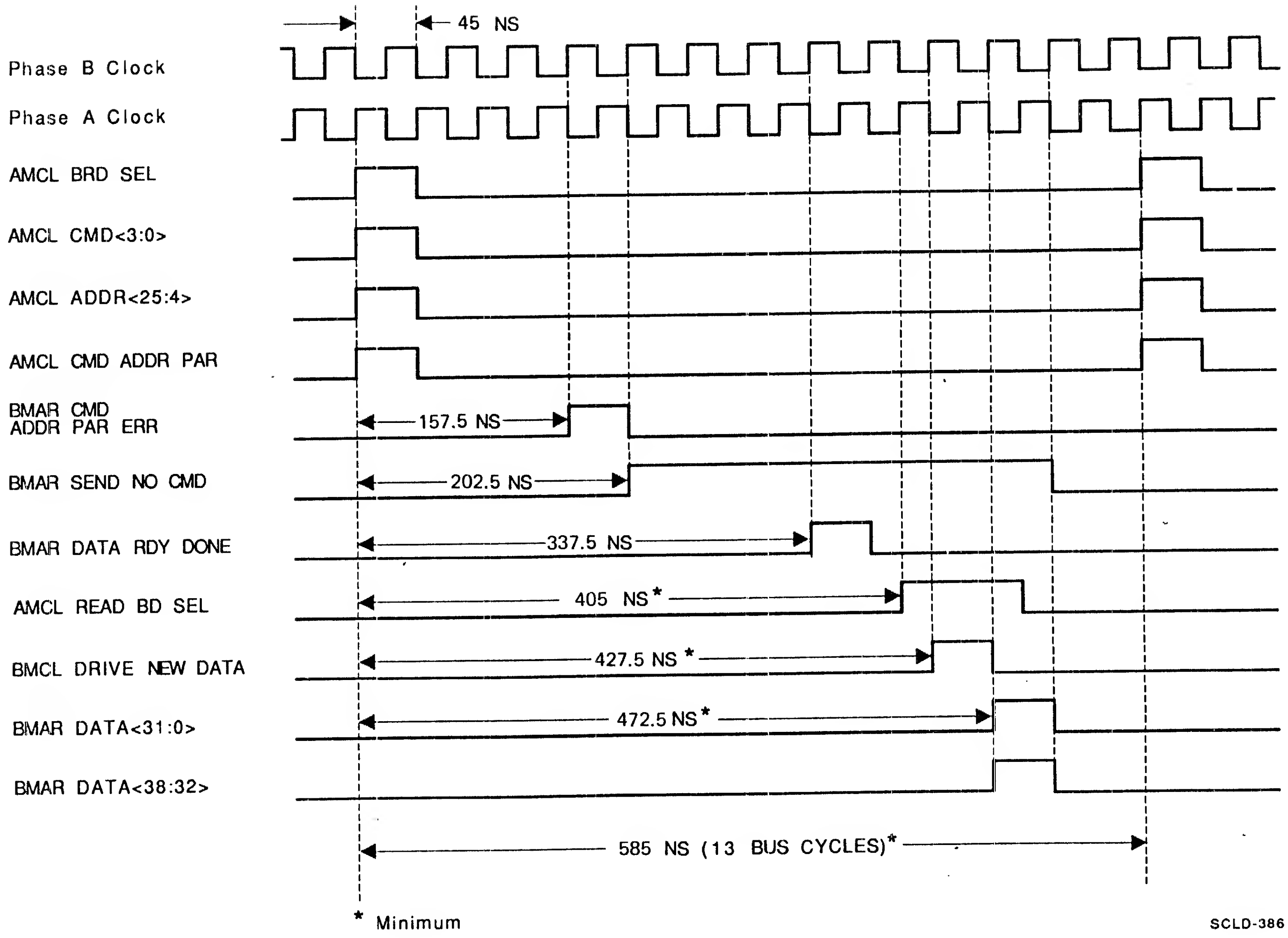


Figure 3-2 Longword Read Timing Diagram

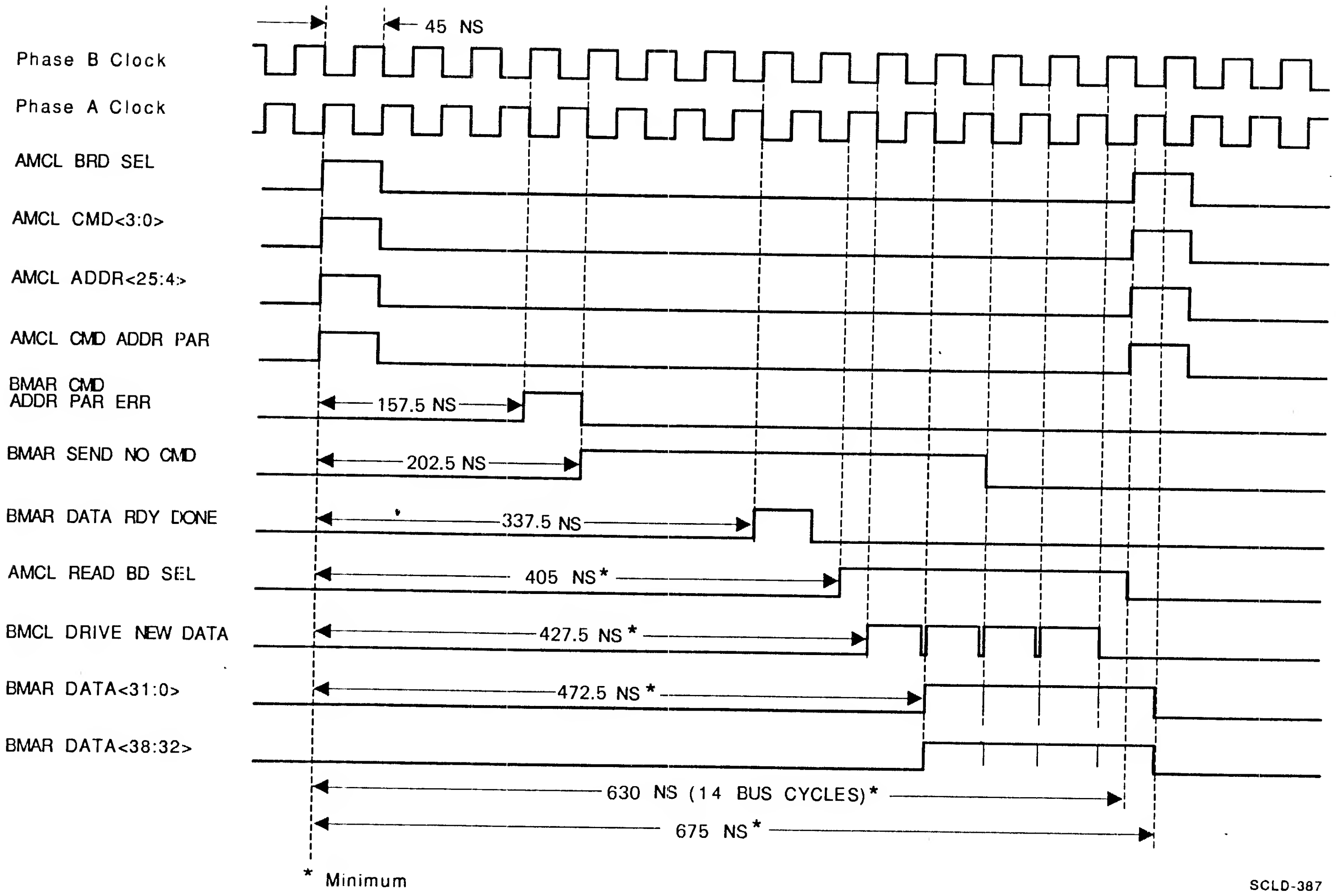


Figure 3-3 Octaword Read Timing Diagram

3.1.3 Longword Read Timing

Figure 3-2 illustrates NAB signal timing for a longword read operation.

In the first bus cycle (command/address cycle), the MCL asserts the board select line for the desired MAR4. It also asserts the command/address and the command/address parity bit. The board select and command/address signals are referenced to the A clock.

If the MAR4 detects a parity error in the command/address, it asserts the command/address parity error bit 157.5 ns from time 0. The command/address parity error bit is referenced to the B clock.

Two hundred two and one-half ns from time 0 the MAR4 asserts BMAR SEND NO CMD. BMAR SEND NO CMD prevents the MCL from issuing any commands to the MAR4 until it has taken the read data. BMAR SEND NO CMD remains asserted until the MCL initiates a transfer of the read data.

Three hundred thirty-seven and one-half ns from time 0 the MAR4 asserts the data-ready-done bit indicating it has completed reading the selected bank and the read data is available. The detection of a command/address parity error has no effect on a read operation in the MAR4. The operation executes in a normal manner.

Sixty-seven and one-half ns after the MCL senses the assertion of the data-ready-done bit, it can assert the MAR4's read-board-select bit (AMCL READ BD SEL). The 67.5 ns is a minimum. The MCL can wait longer if it is busy with another array board.

AMCL READ BD SEL prepares the MAR4 to transfer the read data to the MCL. The MAR4 negates BMAR SEND NO CMD 112.5 ns after receiving AMCL READ BD SEL.

Twenty-two and one-half ns after the assertion of AMCL READ BD SEL (427.5 ns from time 0), the MCL asserts BMCL DRIVE NEW DATA to take the read data from the MAR4. The read data appears on the NAB one bus cycle later.

One hundred twelve and one-half ns after the read data appears on the NAB, the MCL can issue another command to the MAR4. Five hundred eighty-five ns (13 bus cycles) is the minimum time between read operations. Unlike a write operation, consecutive longword reads cannot occur to the four banks. Once a command is issued to read a MAR4 bank, the bank must be read and the data transferred to the MCL before another command can be given to the MAR4.

3.1.4 Octaword Read Timing

Figure 3-3 illustrates NAB signal timing for an octaword read operation.

The functioning of the command/address cycle, the command/address parity error bit, the send-no-command bit, the data-ready-done bit, and the read-board-select bit for an octaword read is identical to their functioning for a longword read. (All but the last paragraph under Section 3.1.3 applies to an octaword read.)

In an octaword read, the MCL holds the MAR4's read-board-select line asserted until the read data from the last bank is on the NAB. AMCL READ BD SEL actually negates while the read data is still on the bus.

As soon as AMCL READ BD SEL negates, the MCL can issue another command to the MAR4. The minimum time between octaword reads is seen to be 630 ns (14 bus cycles)*.

The MCL can assert BMCL DRIVE NEW DATA for four consecutive bus cycles resulting in the read data from the four banks appearing on the NAB in the bus cycles following each drive-new-data pulse. This case is shown in Figure 3-3. (Note that the read-board-select line is negated while the final read data is on the NAB.) The MCL may let more than one cycle occur between the assertions of BMCL DRIVE NEW DATA. In this case, the read data presently existing on the NAB will remain until BMCL DRIVE NEW DATA re-asserts to place new read data on the NAB.

* Note that the MCL can issue a new command relatively sooner after an octaword read operation (630 ns to read four banks) than after a longword read operation (585 ns to read only one bank). During a read operation, the MCL looks for the negation of the send-no-command and the read-board-select lines to know when it can issue another command. In an octaword read, the gating factor is the negation of AMCL READ BD SEL by the MCL which can immediately issue a new command. In a longword read, the gating factor is the negation of BMAR SEND NO CMD by the MAR4. The overhead time required by the MCL to detect the negation of BMAR SEND NO CMD and respond to it accounts for the relative extra time in a longword read operation.

3.2 MAR4 OVERVIEW

Figure 3-4 is an overall block diagram of the MAR4 board. The figure includes all of the NAB signals and illustrates their function on the MAR4 board. The figure divides the MAR4 into six functional areas as listed below:

1. Clock -- The clock logic supplies all the clocks requires by the array board.

2. 4MBARRAY banks -- The 4MBARRAY banks consist of four identical* banks with each bank providing one megabyte of data storage. An array bank contains thirty-nine 256K x 1 DRAM chips resulting in 256K 39-bit locations. A 39-bit location is made up of a 32-bit data longword and seven ECC check bits. The 256K of longword storage in a bank equates to one megabyte of data storage. The four banks provide the array board with a total storage capacity of four megabytes. The array banks use a common I/O path with the data and check bits passing through bi-directional latches within the banks.

3. Input parser -- The input parser processes the input command/address and controls many of the board functions. It also performs a parity check on the command/address and informs the MCL of any parity error.

4. ECC/DPARITY -- The ECC/DPARITY logic generates partial check bits on the input write data. It also performs a parity check on the input data and two data information bits (bad data bit and write enable bit). It generates an INT BAD DATA bit that is incorporated into the ECC check bits outside the ECC/DPARITY logic.

5. Data-output control -- The data-output control controls the read data flow out of the array board.

6. Refresh -- The refresh logic performs a periodic refresh of the arrays when they are not busy doing a read or a write.

The six functional areas are shown in the block diagram along with the major signals that connect one area to another. The engineering logic prints are grouped into the same six functional areas.

* Routing of two MAR4 signals (CLR REF REQ and MCL COLD START) is not identical in all four banks. This minor difference is described in Section 3.3.2.7. For functional purposes the banks can be considered identical.

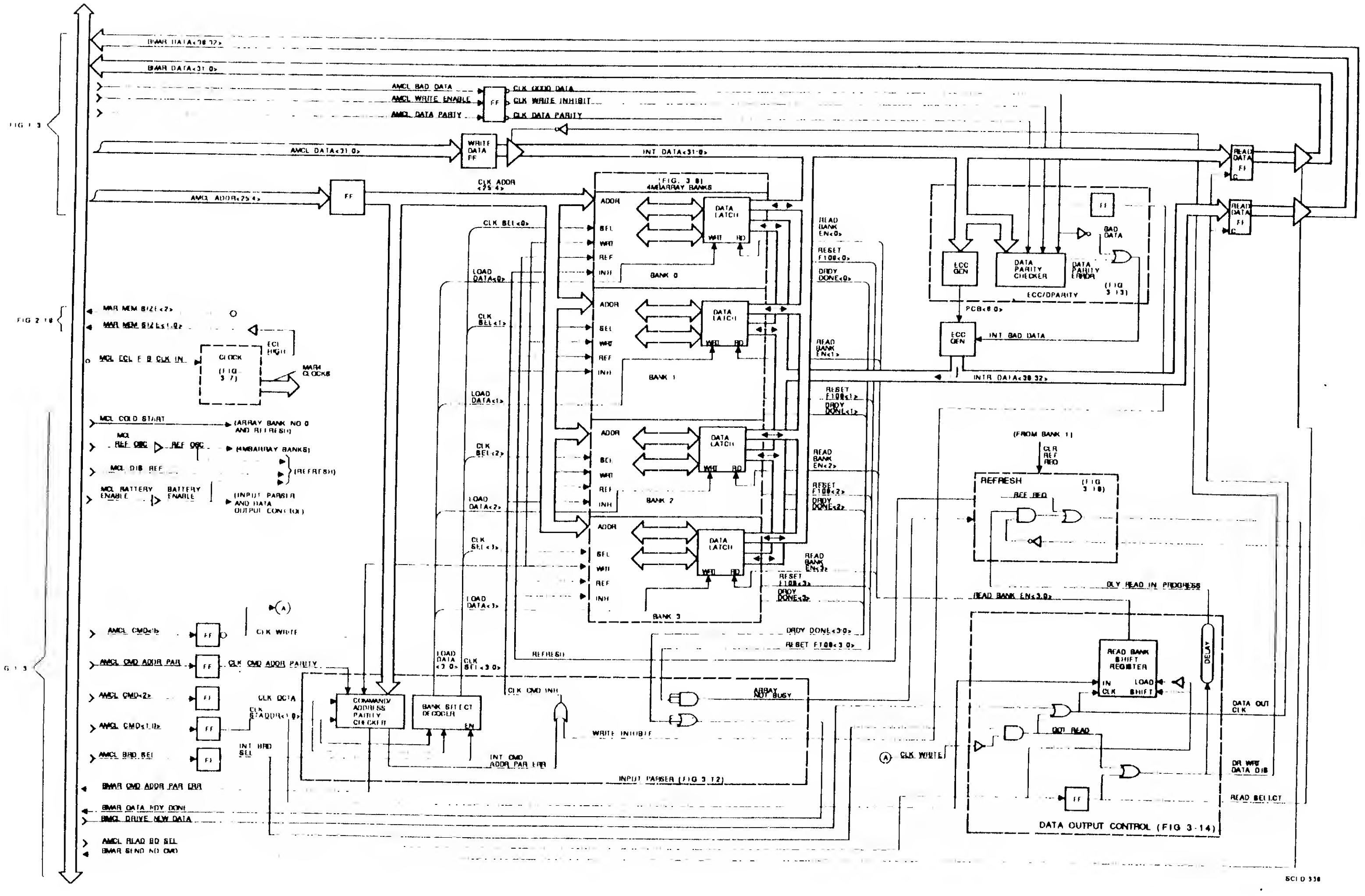


Figure 3-4 MAR4 Block Diagram

Figures 3-5 and 3-6 are flow diagrams illustrating the three command operations that can be executed by a MAR4 (longword write, longword read, octaword read). Refer to the block diagram (Figure 3-4) and the flow diagrams (Figures 3-5 and 3-6) during the following discussion.

3.2.1 Write Operation (Figures 3-4 and 3-5)

There are eight BMAR SEND NO CMD signal lines on the NAB with one line connected to each array board. An array board will assert its BMAR SEND NO CMD line when it is not able to accept a command/address from the MCL. An array board cannot accept a command/address when it is doing a read operation or a refresh. When doing a write operation, BMAR SEND NO CMD is not asserted because only the bank being written is busy. Another write could be initiated to one of the other three banks.

Before sending a command/address to a MAR4, the MCL checks the MAR4's BMAR SEND NO CMD line. If BMAR SEND NO CMD is false, the MCL selects the MAR4 by asserting AMCL BRD SEL on the MAR4's board select line.

There are eight AMCL BRD SEL lines on the NAB with one line connected to each array board. The MCL selects a board by asserting AMCL BRD SEL on the select line connected to that board. AMCL BRD SEL is clocked into a flip-flop to become INT BRD SEL. INT BRD SEL is applied to the input parser where it enables a bank-select decoder to process the input command.

The MCL also asserts a 4-bit command (AMCL CMD<3:0>), an address (AMCL ADDR<25:4>), and a command/address parity bit (AMCL CMD ADDR PAR) on the NAB. The command/address and its associated parity bit are clocked into flip-flops to become CLK WRITE, CLK OCTA, CLK STADDR<1:0>, CLK ADDR<25:4>, and CLK CMD ADDR PAR. The CLK WRITE and CLK OCTA bits of the input command specify the type of operation to be performed (write, read, octaword read). The command write bit (AMCL CMD<3>) is negated for a write operation. Signal inversion results in CLK WRITE being asserted. The CLK STADDR<1:0> bits (starting address) are a two-bit code specifying the bank array to be accessed for the write. These signals are applied to a parity checker in the input parser.

If the parity checker detects a command/address parity error, it asserts INT CMD ADDR PAR ERR causing CLK CMD INH to assert to all the array banks. CLK CMD INH inhibits the CAS (column address strobe) sequence in all four banks. With no CAS sequence, data will not be written into an array. All other actions of the write operation sequence will occur normally.

In addition to inhibiting the CAS sequence, a command/address parity error will assert BMAR CMD ADDR PAR ERR back to the MCL indicating that a command/address parity error has occurred. The MCL will abort the operation and assert a memory interrupt on the NMI.

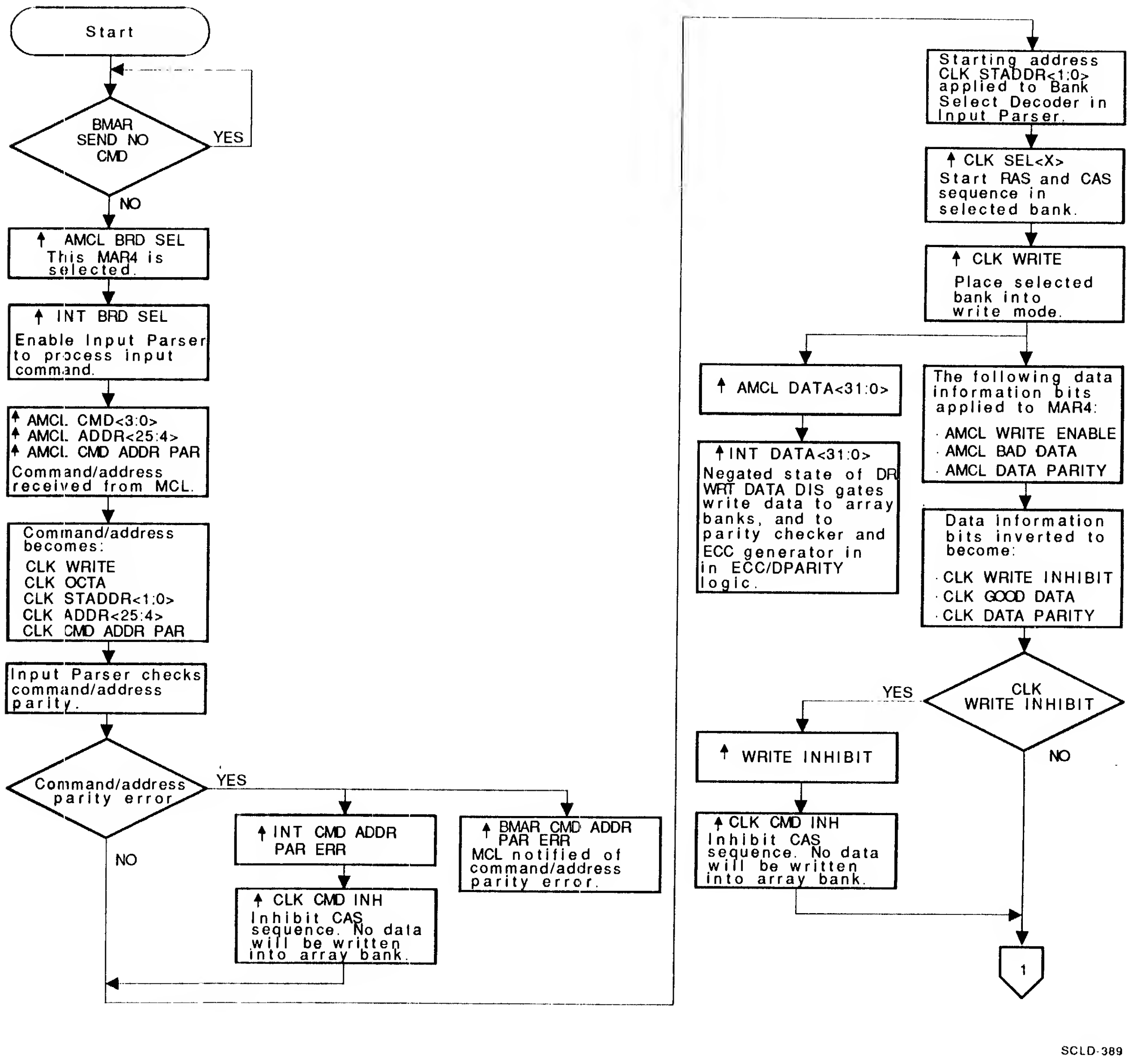

Start
BMAR SEND NO CMD
YES
NO
↑ AMCL BRD SEL This MAR4 is selected.
↑ INT BRD SEL Enable Input Parser to process input command.
↑ AMCL CMD<3:0>
↑ AMCL ADDR<25:4>
↑ AMCL CMD ADDR PAR
Command/address received from MCL.
Command/address becomes:
CLK WRITE
CLK OCTA
CLK STADDR<1:0>
CLK ADDR<25:4>
CLK CMD ADDR PAR
Input Parser checks command/address parity.
Command/address parity error
YES
NO
↑ INT CMD ADDR PAR ERR
↑ CLK CMD INH Inhibit CAS sequence. No data will be written into array bank.
↑ BMAR CMD ADDR PAR ERR MCL notified of command/address parity error.
Starting address CLK STADDR<1:0> applied to Bank Select Decoder in Input Parser.
↑ CLK SEL<X> Start RAS and CAS sequence in selected bank.
↑ CLK WRITE Place selected bank into write mode.
↑ AMCL DATA<31:0>
↑ INT DATA<31:0> Negated state of DR WRT DATA DIS gates write data to array banks, and to parity checker and ECC generator in in ECC/DPARITY logic.
The following data information bits applied to MAR4:
· AMCL WRITE ENABLE
· AMCL BAD DATA
· AMCL DATA PARITY
Data information bits inverted to become:
· CLK WRITE INHIBIT
· CLK GOOD DATA
· CLK DATA PARITY
CLK WRITE INHIBIT
YES
NO
↑ WRITE INHIBIT
↑ CLK CMD INH Inhibit CAS sequence. No data will be written into array bank.
1
SCLD-389

Figure 3-5 Write Flow Diagram (Sheet 1 of 2)

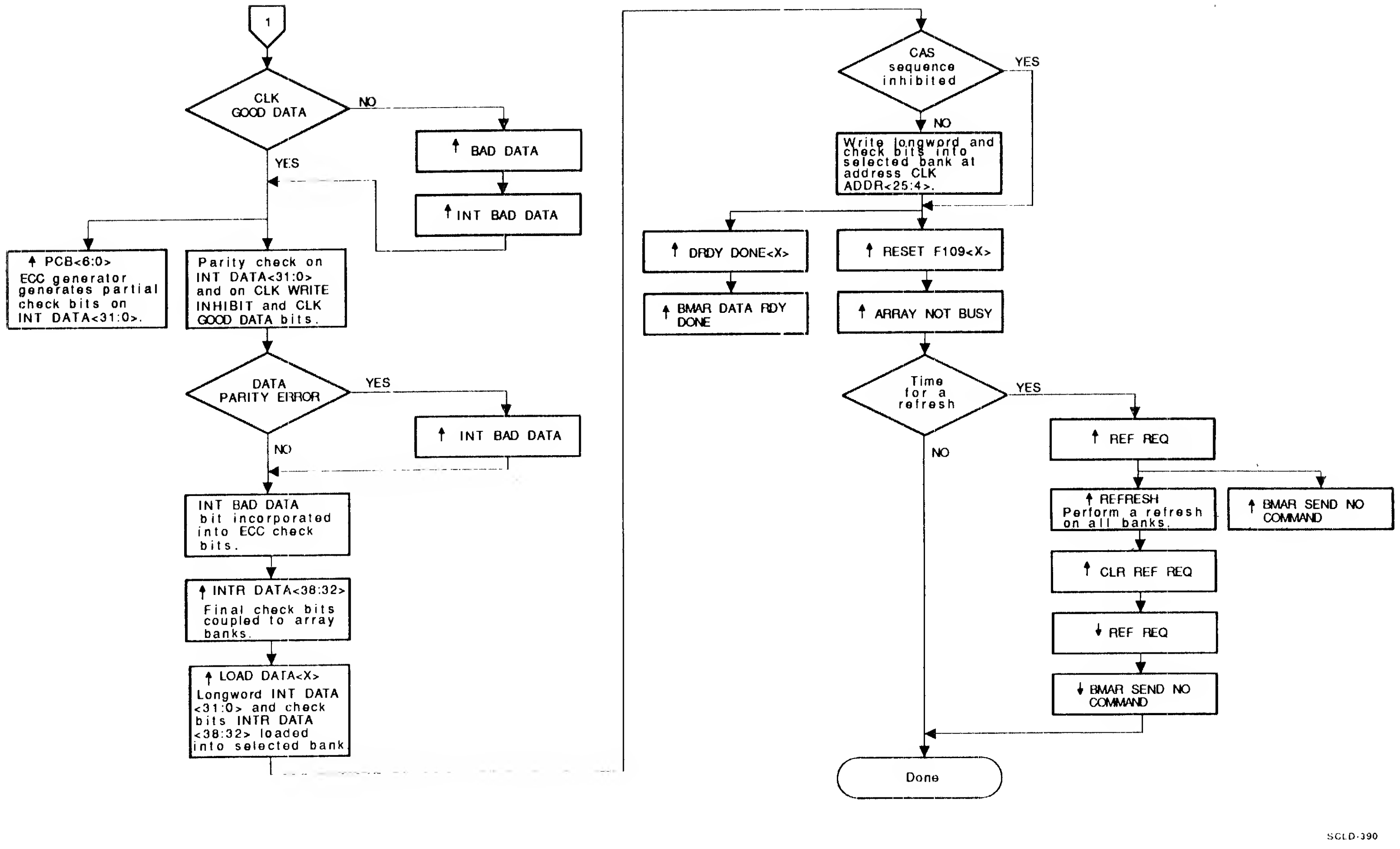

1
CLK GOOD DATA
NO
YES
↑ BAD DATA
↑ INT BAD DATA
↑ PCB<6:0> ECC generator generates partial check bits on INT DATA<31:0>.
Parity check on INT DATA<31:0> and on CLK WRITE INHIBIT and CLK GOOD DATA bits.
DATA PARITY ERROR
YES
↑ INT BAD DATA
NO
INT BAD DATA bit incorporated into ECC check bits.
↑ INTR DATA<38:32> Final check bits coupled to array banks.
↑ LOAD DATA<X> Longword INT DATA <31:0> and check bits INTR DATA <38:32> loaded into selected bank.
CAS sequence inhibited
YES
NO
Write longword and check bits into selected bank at address CLK ADDR<25:4>.
↑ DRDY DONE<X>
↑ BMAR DATA RDY DONE
↑ RESET F109<X>
↑ ARRAY NOT BUSY
Time for a refresh
YES
NO
↑ REF REQ
↑ REFRESH Perform a refresh on all banks.
↑ BMAR SEND NO COMMAND
↑ CLR REF REQ
↓ REF REQ
↓ BMAR SEND NO COMMAND
Done
SCLD-390

Figure 3-5 Write Flow Diagram (Sheet 2 of 2)

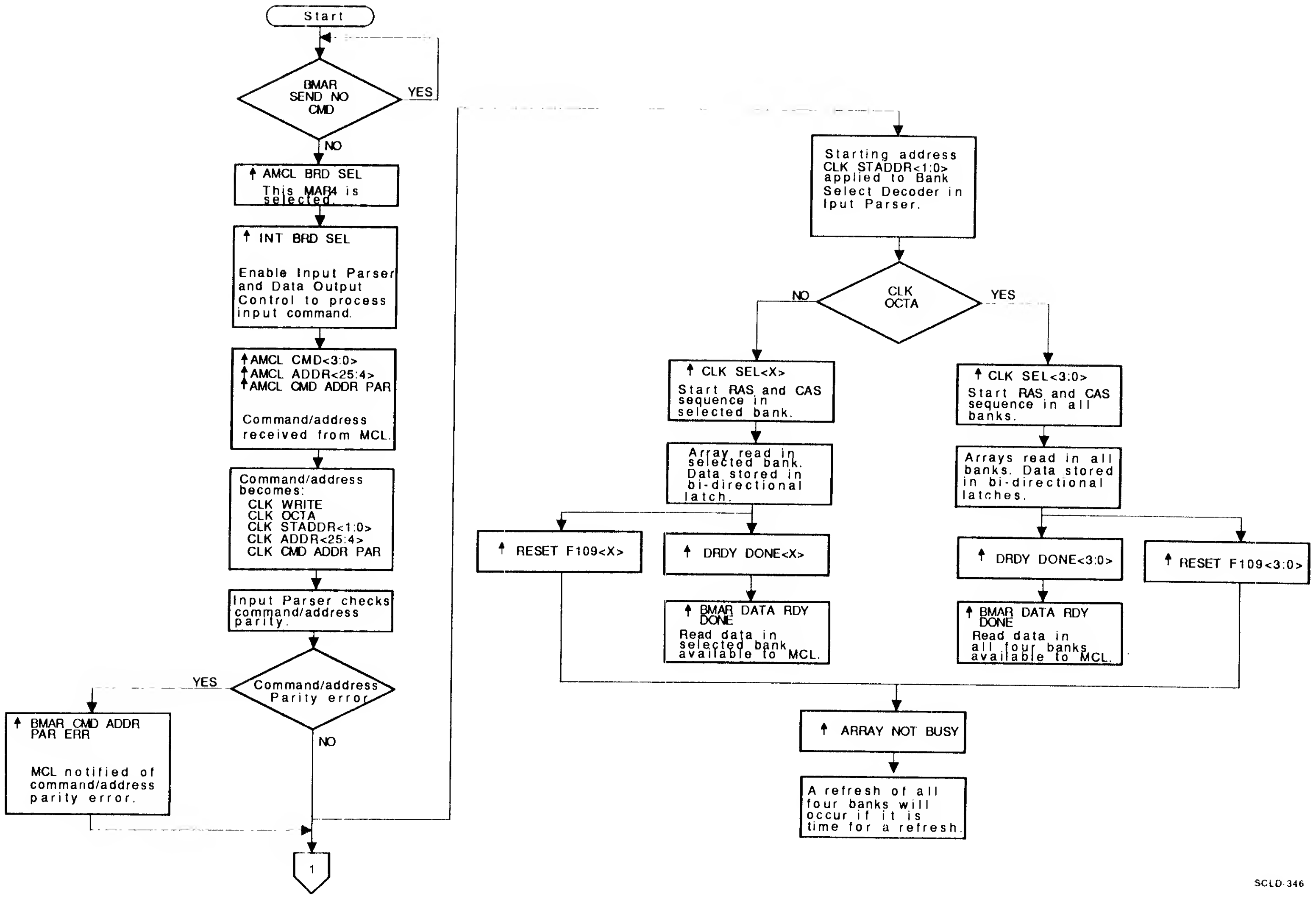
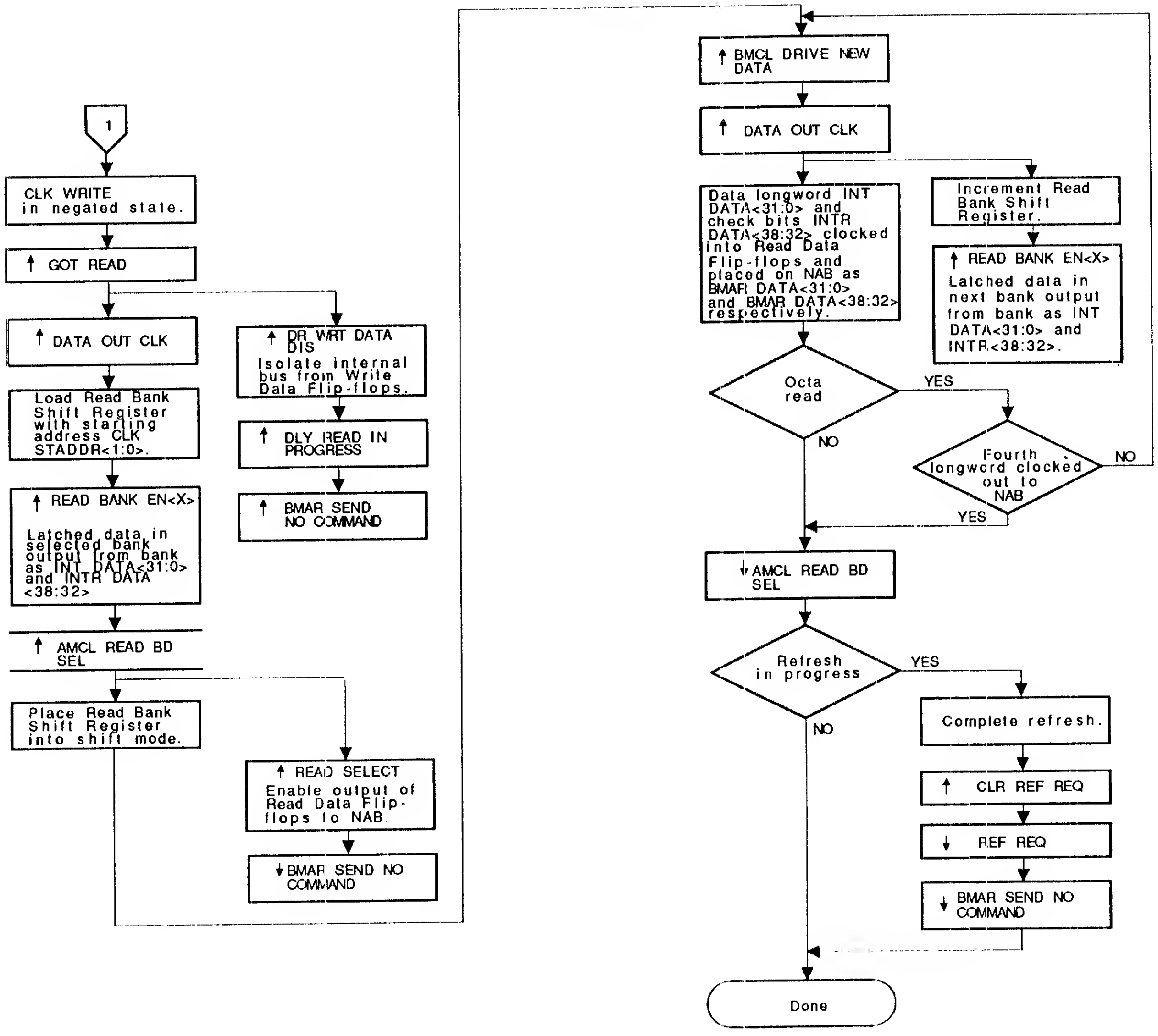

Start
BMAR SEND NO CMD
YES
NO
↑ AMCL BRD SEL
This MAR4 is selected.
↑ INT BRD SEL
Enable Input Parser and Data Output Control to process input command.
↑ AMCL CMD<3:0>
↑ AMCL ADDR<25:4>
↑ AMCL CMD ADDR PAR
Command/address received from MCL.
Command/address becomes:
CLK WRITE
CLK OCTA
CLK STADDR<1:0>
CLK ADDR<25:4>
CLK CMD ADDR PAR
Input Parser checks command/address parity.
Command/address Parity error
YES
NO
↑ BMAR CMD ADDR PAR ERR
MCL notified of command/address parity error.
1
Starting address CLK STADDR<1:0> applied to Bank Select Decoder in Iput Parser.
CLK OCTA
NO
YES
↑ CLK SEL<X>
Start RAS and CAS sequence in selected bank.
Array read in selected bank. Data stored in bi-directional latch.
↑ RESET F109<X>
↑ DRDY DONE<X>
↑ BMAR DATA RDY DONE
Read data in selected bank available to MCL.
↑ CLK SEL<3:0>
Start RAS and CAS sequence in all banks.
Arrays read in all banks. Data stored in bi-directional latches.
↑ DRDY DONE<3:0>
↑ RESET F109<3:0>
↑ BMAR DATA RDY DONE
Read data in all four banks available to MCL.
↑ ARRAY NOT BUSY
A refresh of all four banks will occur if it is time for a refresh.
SCLD-346

Figure 3-6 Read Flow Diagram (Sheet 1 of 2)



1
CLK WRITE in negated state.
↑ GOT READ
↑ DATA OUT CLK
Load Read Bank Shift Register with starting address CLK STADDR<1:0>.
↑ READ BANK EN<X> Latched data in selected bank output from bank as INT DATA<31:0> and INTR DATA <38:32>
↑ AMCL READ BD SEL
Place Read Bank Shift Register into shift mode.
↑ DR WRT DATA DIS Isolate internal bus from Write Data Flip-flops.
↑ DLY READ IN PROGRESS
↑ BMAR SEND NO COMMAND
↑ READ SELECT Enable output of Read Data Flip-flops to NAB.
↓ BMAR SEND NO COMMAND
↑ BMCL DRIVE NEW DATA
↑ DATA OUT CLK
Data longword INT DATA<31:0> and check bits INTR DATA<38:32> clocked into Read Data Flip-flops and placed on NAB as BMAR DATA<31:0> and BMAR DATA<38:32> respectively.
Increment Read Bank Shift Register.
↑ READ BANK EN<X> Latched data in next bank output from bank as INT DATA<31:0> and INTR<38:32>.
Octa read
YES
NO
Fourth longword clocked out to NAB
NO
YES
↓ AMCL READ BD SEL
Refresh in progress
YES
NO
Complete refresh.
↑ CLR REF REQ
↓ REF REQ
↓ BMAR SEND NO COMMAND
Done

SCLD-347

Figure 3-6 Read Flow Diagram (Sheet 2 of 2)

Note that the command/address parity check is independent of board selection. All array boards on the NAB make a parity check of all command/addresses placed on the NAB. However, only the first array slot (slot number 0) is connected to the BMAR CMD ADDR PAR ERR line on the NAB. Hence, the array board in slot 0 does the command/address parity checking for all command/addresses on the NAB regardless of which array board is selected for the command/address. It follows from this that slot 0 must always be used.

Address CLK ADDR<25:4> is applied to the four array banks. The address specifies the location in the array that is to be accessed*.

Starting address CLK STADDR<1:0> is applied to the bank-select decoder in the input parser. The decoder asserts CLK SEL<X> where X is the number of the selected bank. CLK SEL<X> initiates the RAS and CAS strobing sequence in the bank. The strobing sequence will access the bank array at the address specified by CLK ADDR<25:4>. As noted earlier, if a command/address parity error occurred, the CAS strobing sequence will not execute.

Command bit CLK WRITE is true and places the selected bank into the write mode. CLK WRITE is applied to all four banks but only the selected bank has its strobing sequence enabled.

The MCL places the write data (AMCL DATA<31:0>), a write enable signal (AMCL WRITE ENABLE), a bad data signal (AMCL BAD DATA), and a data parity bit (AMCL DATA PARITY) on the NAB.

The write data bits are clocked into write-data flip-flops. The outputs of the flip-flops are enabled onto the MAR4 internal bus by the negated state of DR WRT DATA DIS where they become INT DATA<31:0>. INT DATA<31:0> is applied to bi-directional data latches in all four array banks, and to an ECC generator and a data parity checker in the ECC/DPARITY logic.

AMCL WRITE ENABLE is true if the write data is to be written into the MAR4. A write enable signal is necessary because there are times when some longwords in an octaword write operation are not written (Section 1.4.3). When less than four longwords are to be written, the unwanted write data is masked with the write enable signal.

AMCL WRITE ENABLE is inverted to become CLK WRITE INHIBIT. If AMCL WRITE ENABLE is false (write data is not to be written), CLK WRITE INHIBIT will be true. CLK WRITE INHIBIT is applied to the ECC/DPARITY logic where it asserts WRITE INHIBIT. WRITE INHIBIT is applied to the input parser where it asserts CLK CMD INH. CLK CMD

* The 4MBARRAY banks do not use the four most significant bits of the address (CLK ADDR<25:22>). (See Section 3.3.2.3.)

INH is applied to all the array banks where it inhibits the CAS sequence in the selected array thereby preventing the write data from being written.

AMCL BAD DATA is asserted by the MCL when it has detected a parity error or missing write data. The bad data is written so that it can be analyzed later by maintenance diagnostics.

AMCL BAD DATA is inverted to become CLK GOOD DATA. If AMCL BAD DATA is true (write data is bad), CLK GOOD DATA will be false. CLK GOOD DATA is applied to the ECC/DPARITY logic. The negated state of CLK GOOD DATA causes BAD DATA to assert. BAD DATA asserts INT BAD DATA which is applied to a portion of the ECC generator that is external to the ECC/DPARITY logic.

AMCL DATA PARITY is the parity bit on the write data and on the bad data and write enable bits. AMCL DATA PARITY is inverted to become CLK DATA PARITY. CLK DATA PARITY is applied to a data parity checker in the ECC/DPARITY logic. Here parity is checked on the input data and on the write enable and the bad data bits. If an error is detected, DATA PARITY ERROR asserts which in turn asserts INT BAD DATA (if not already asserted by BAD DATA).

The write data (INT DATA<31:0>) is applied to an ECC generator where seven partial* check bits (PCB<6:0>) are generated. The partial check bits are applied to more ECC generator logic (outside the ECC/DPARITY logic) where the INT BAD DATA bit is incorporated to generate the final check bits (INTR DATA<38:32>).

The generated check bits are unique for the input data from which they are derived. With INT BAD DATA being one of the inputs, the final check bits reflect the state of the INT BAD DATA bit (see Section 2.8.2). An analysis of the check bits may show the data as being bad (INT BAD DATA as being true) but it will not catagorize the error as being a parity error (DATA PARITY ERROR true), bad data from the MCL (BAD DATA true), or both.

The final check bits (INTR DATA<38:32>) are applied to bi-directional latches in the four array banks along with the write data.

LOAD DATA<X> is asserted by the bank-select decoder and enables the data latch in the selected bank to pass the input data to the bank array+. The RAS and CAS strobing sequence in the selected

* The check bits are partial in a preliminary sense, -- that is, more processing is to be done before they are final. All seven check bits exist in the "partial" state.

+ LOAD DATA<X> enables only the longword to pass to the array. A delayed signal (DLY LOAD DATA<X>) is issued by the bank-select decoder to enable the check bits to pass to the array (see Section 3.3.2.2 under 4MBARRAY Banks).

bank will write the data into the array. If CLK CMD INH is true (due to a command/address parity error or an asserted write inhibit signal), the CAS strobing sequence is inhibited and the data is not written into the array.

After the selected bank writes the 32 data bits and 7 check bits into the array, it asserts DRDY DONE<X> and RESET F109<X> to the input parser. DRDY DONE<X> asserts the MAR4's BMAR DATA RDY DONE line on the NAB. There are eight BMAR DATA RDY DONE lines on the NAB with one line connected to each array board. The assertion of BMAR DATA RDY DONE indicates to the MCL that the write operation commanded for this MAR4 has been completed.

RESET F109<X> is applied to the input parser where it is ANDed with RESET F109 signals from the other three banks to assert ARRAY NOT BUSY. (The RESET F109 signals from the other arrays are true.) ARRAY NOT BUSY is applied to the refresh logic to indicate that all the banks are free. If the refresh logic deems it time for a refresh cycle, it asserts REF REQ which in turn asserts BMAR SEND NO COMMAND on the NAB and REFRESH to all the array banks. REFRESH initiates a refresh cycle in all four array banks. When the refresh cycle is finished, bank 1 asserts CLR REF REQ to the refresh logic which then negates REF REQ and BMAR SEND NO COMMAND. All the array banks refresh simultaneously hence only one bank need notify the refresh logic of the completion of the refresh cycle.

With BMAR SEND NO COMMAND negated, the MCL is allowed to send another command to the MAR4 board.

3.2.2 Read Operation (Figures 3-4 and 3-6)

In many respects, a read operation starts out like a write operation. Before sending a command/address to a MAR4, the MCL checks the MAR4's BMAR SEND NO CMD line. If BMAR SEND NO CMD is false, the MCL selects the MAR4 by asserting AMCL BRD SEL on the MAR4's board select line. AMCL BRD SEL becomes INT BRD SEL and is applied to the input parser and the data-output control where it enables the logic to process the input command.

The MCL asserts the 4-bit command, the address, and the command/address parity bit on the NAB. The command/address and the command/address parity bit become CLK WRITE (false for a read operation), CLK OCTA, CLK STADDR<1:0>, CLK ADDR<25:4>, and CLK CMD ADDR PAR. These signals are applied to a parity checker in the input parser.

If a command/address parity error is detected, BMAR CMD ADDR PAR ERR is asserted back to the MCL indicating that a command/address parity error has occurred. After notifying the MCL that a command/address parity error has occurred, the MAR4 continues to do a normal read operation. This is in contrast to a write operation where the array CAS sequence is inhibited.

As discussed under Section 3.2.1 (Write Operation), only array slot number 1 is connected to the BMAR CMD ADDR PAR ERR line on the NAB. Hence, the array board in slot 1 does the command/address parity check for all command/addresses on the NAB regardless of which array board is selected for the command/address. It follows from this that slot 1 must always be used.

Address CLK ADDR<25:4> is applied to the four array banks. The address specifies the location in the array that is to be accessed*.

Starting address CLK STADDR<1:0> and octa-read bit CLK OCTA are applied to the bank-select decoder in the input parser. If CLK OCTA is false (longword read), the bank-select decoder outputs CLK SEL<X> where X is the number of the selected bank. CLK SEL<X> initiates the RAS and CAS strobing sequence in the selected bank. With CLK WRITE false, the selected bank is placed into the read mode. The strobing sequence functions to read the array at the address specified by CLK ADDR<25:4> and place the data into a bi-directional latch in the bank. The array bank then asserts DRDY DONE<X> and RESET F109<X> indicating that the array has been read.

DRDY DONE<X> is applied to the input parser where it asserts the MAR4's BMAR DATA RDY DONE line on the NAB. The assertion of BMAR DATA RDY DONE informs the MCL that the array has been read and the read data is available.

RESET F109<X> is applied to the input parser where it asserts ARRAY NOT BUSY to the refresh logic. ARRAY NOT BUSY indicates to the refresh logic that the arrays are free and a refresh can be executed if one is needed.

If CLK OCTA is true (octaword read), the bank-select decoder outputs CLK SEL<3:0> to all four array banks. CLK SEL<3:0> initiates the RAS and CAS strobing sequence in all of the array banks. With CLK WRITE false, the four banks are read at the CLK ADDR<25:4> address and the data placed into the banks' bi-directional latches. After placing the read data into the bi-directional latches, each bank asserts its RESET F109 and DRDY DONE signals indicating that its array has been read.

A DRDY DONE signal from any bank asserts the MAR's BMAR DATA RDY DONE line on the NAB. The assertion of BMAR DATA RDY DONE informs the MCL that the array banks have been read and the read data is available.

The RESET F109 signals are applied to the input parser where they are ANDed to assert ARRAY NOT BUSY to the refresh logic. ARRAY NOT BUSY indicates to the refresh logic that the arrays are free and a refresh can be executed if one is needed.

* The 4MBARRAY banks do not use the four most significant bits of the address (CLK ADDR<25:22>). (See Section 3.3.2.3.)

The data-output control contains a read-bank shift register which normally is in the "load" mode of operation. With CLK WRITE false and the MAR4 board selected (INT BRD SEL true), GOT READ asserts which in turn asserts DATA OUT CLK and DR WRT DATA DIS. DATA OUT CLK clocks the shift register thereby loading in starting address CLK STADDR<1:0>. CLK STADDR<1:0> is decoded and causes the register to output READ BANK EN<X> where X is the selected bank. Bank X is the bank to be read in a longword read or the first bank to be read in an octaword read. READ BANK EN<X> is applied to the selected bank where it couples the data longword and check bits from the bi-directional latch to the internal bus as INT DATA<31:0> and INTR DATA<38:32> respectively.

DR WRT DATA DIS isolates the internal bus from the write-data flip-flops to ensure that the internal bus is driven only by the array outputs.

DR WRT DATA DIS also asserts DLY READ IN PROGRESS to the refresh logic causing BMAR SEND NO COMMAND to assert. BMAR SEND NO COMMAND inhibits the MCL from sending any new commands to the MAR4 until the MCL takes the read data.

The MCL, sensing the true state of BMAR DATA RDY DONE, asserts AMCL READ BD SEL to select the MAR4 and prepare it to transfer the read data. There is no time requirement in which the MCL must take the read data. After sensing the assertion of BMAR DATA RDY DONE, the MCL is free to assert AMCL READ BD SEL whenever it is ready.

There are eight AMCL READ BD SEL lines on the NAB, with one line connected to each array board. The MCL selects the MAR4 board by asserting AMCL READ BD SEL on the line connected to it.

AMCL READ BD SEL is applied to the data-output control where it prepares the MAR4 for the transfer of read data by:

- Placing the read-bank shift register into the "shift" mode of operation.
- Asserting READ SELECT to enable the output of the read-data flip-flops to the NAB.
- Taking control (via READ SELECT) of the assertion of DR WRT DATA DIS thereby making the internal bus-to-NAB data path a function of AMCL READ BD SEL.
- Negating BMAR SEND NO COMMAND (via READ SELECT) to allow the MCL to issue a new command to the MAR4 board after it takes the read data.

The MCL then asserts BMCL DRIVE NEW DATA to clock the read data from the MAR4 board out to the NAB. BMCL DRIVE NEW DATA is applied to the data-output control where it asserts DATA OUT CLK. DATA OUT CLK clocks data longword INT DATA<31:0> and check bits INTR DATA<38:32> from the internal bus into the read-data flip-flops.

The flip-flops output the data to the NAB as BMAR DATA<31:0> and BMAR DATA<38:32> respectively.

DATA OUT CLK also clocks the read-bank shift register (now in shift mode) causing the READ BANK EN<X> output to increment (used to read next bank in an octaword read operation).

If this is a longword read operation, the MCL will negate AMCL READ BD SEL to complete its operation. If no refresh cycle is in progress, the MAR4 is ready to accept a new command from the MCL.

If a refresh cycle is in progress, the MAR4 board cannot receive any new commands until the refresh is completed. When the refresh is completed, array bank 1 will assert CLR REF REQ to the refresh logic which will in turn negate REF REQ and then BMAR SEND NO COMMAND. The negation of BMAR SEND NO COMMAND allows the MCL to issue a new command to the MAR4 board.

If an octaread operation is being executed, the MCL will re-assert BMCL DRIVE NEW DATA to take the second longword from the MAR4. BMCL DRIVE NEW DATA re-asserts DATA OUT CLK which clocks the data off the internal bus into the read-data flip-flops. The incremented READ BANK EN<X> has coupled the read data from the next array bank to the internal bus; hence, it is the second data longword and check bits that are clocked into the flip-flops. The flip-flop outputs appear on the NAB as BMAR DATA<31:0> and BMAR DATA<38:32> (READ SELECT still true).

DATA OUT CLK again increments the read-bank shift register so that READ BANK EN<X> points to the next array bank. The process repeats itself until all four array banks have been read. The MCL then negates AMCL READ BD SEL to de-select the MAR4 board and end the read operation.

If a refresh is in progress, it must complete before BMAR SEND NO CMD is negated and the MCL can issue a new command to the MAR4.

3.3 MAR4 DETAILED DESCRIPTIONS

Detailed descriptions are given of the six functional areas defined in Section 3.2. The descriptions are confined to the area being described. Therefore, it is important that the reader has read and understands the MAR4 overview given in Section 3.2 in order to understand how the signals in the functional areas relate to the overall operation of the MAR4.

A block diagram is provided for each functional area showing input and output signals. These signals are tied to the other block diagrams by reference figure numbers. An exception to this is the Clock Logic Block Diagram where the clock destinations are given but not figure numbers. This is because timing clocks are not included in the detailed description block diagrams.

3.3.1 Clock Logic

Refer to Figure 3-7 (Clock Logic Block Diagram) and Figure 3-8 (Clock Timing Diagram) throughout the following discussion.

The clock logic receives a free running, non-single-stepped, phase B clock (MCL ECL F B CLK IN) from the MCL via the NAB bus. Six other clocks are derived from MCL ECL F B IN and are distributed throughout the MAR4 board. Five of the clocks are generated in the clock logic while the sixth is developed in the data-output control. The sixth clock is discussed in this section so that its relationship to the other clocks can be described. All six clocks are listed below.

- ECL CLK
- TTL CLK BM
- TTL CLK BUS
- TTL CLK LATCH
- ECL SEL LATCH
- DRDY SNC CLK*

ECL CLK, TTL CLK BM, and TTL CLK BUS are identical to input clock MCL ECL F B CLK IN.

MCL ECL F B CLK IN sets a CLK latch flip-flop asserting TTL CLK LATCH. Twenty-three ns later the flip-flop is reset. This results in a TTL CLK LATCH clock with the leading edge essentially in phase with MCL ECL F B CLK IN but with an asymmetrical waveshape of 23 ns and 22 ns.

MCL ECL F B CLK IN is delayed 24 ns and then sets a SEL Latch flip-flop asserting ECL SEL LATCH. Eighteen ns later the flip-flop is reset. This results in an ECL SEL LATCH clock delayed 24 ns with respect to MCL ECL F B CLK IN and with an asymmetrical waveshape of 18 ns and 27 ns.

ECL HIGH is a positive voltage used to supply the MAR4 memory size code to the MCL.

The ECL CLK is sent to the data-output control where it is delayed 20 ns and then used to set a DRDY SNC flip-flop. The flip-flop asserts the DRDY SNC CLK clock which is fed back through a 9 ns delay to reset the flip-flop. This results in a DRDY SNC CLK clock with its leading edge delayed 20 ns with respect to the phase B clock and with an asymmetrical waveshape of 9 ns and 36 ns.

DRDY SNC CLK is closer to being a phase A clock than a phase B clock (see Section 3.1.1 for clock phases). This is due to the 20 ns delay of the leading edge of DRDY SNC CLK with respect to the

* Generated in the data-output control.

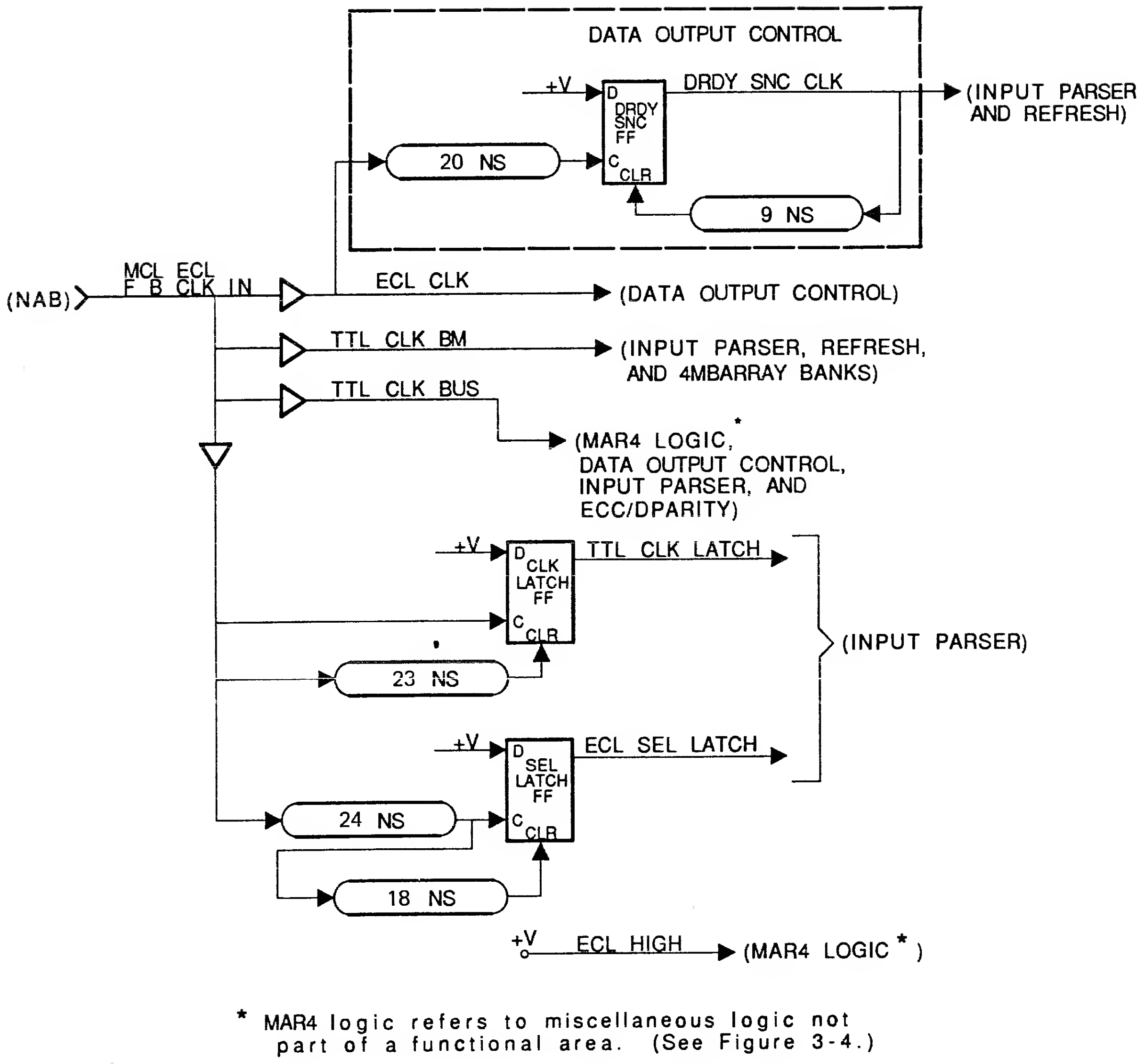


Figure 3-7 Clock Logic Block Diagram

phase B clock. Hence DRDY SNC CLK is used to clock all data going to the MCL (the MCL operates on the phase A clock). The MAR4-to-MCL signals clocked by DRDY SNC CLK are:

- In the data-output control -- DATA OUT CLK is generated by the DRDY SNC CLK circuit and is in phase with DRDY SNC CLK. DATA OUT CLK clocks the read-data flip-flops which place the data longword (BMAR DATA<31:0>) and check bits (BMAR DATA<38:32:) on the NAB.

- In the input parser -- BMAR CMD ADDR PAR ERR BMAR DATA RDY DONE

- In the refresh logic -- BMAR SEND NO CMD

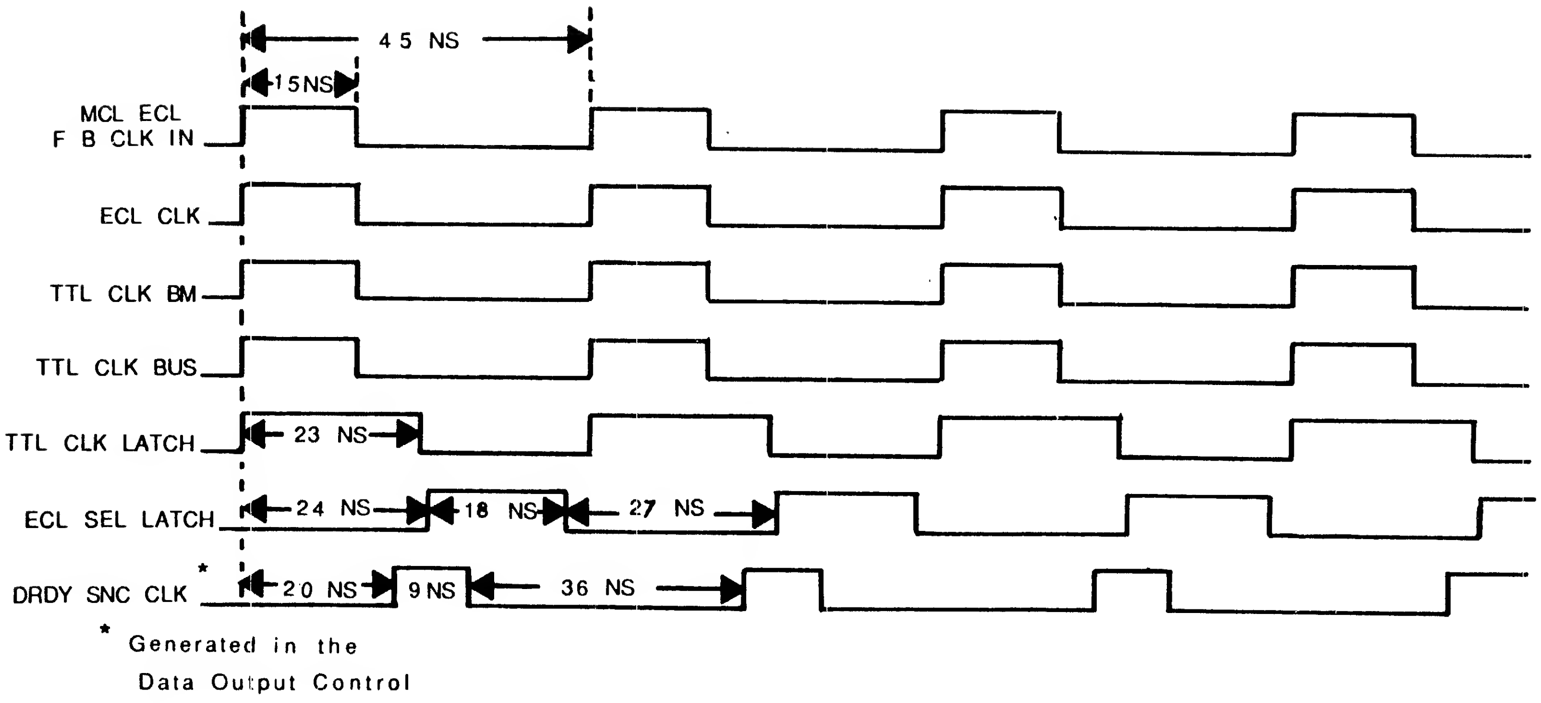


Figure 3-8 Clock Timing Diagram

Figure 3-7 identifies the destinations of all the clock signals. Table 3-2 lists the destinations by functional area.

Table 3-2 Clock Distribution

Functional Area	Clock(s) Received
MAR4 logic *	TTL CLK BUS ECL HIGH +
4MBARRAY banks	TTL CLK BM
Input parser	TTL CLK BM TTL CLK BUS TTL CLK LATCH ECL SEL LATCH DRDY SNC CLK
Data-output control	ECL CLK TTL CLK BUS
ECC/DPARITY	TTL CLK BUS
Refresh	TTL CLK BM DRDY SNC CLK

* MAR4 logic refers to miscellaneous logic not part of a functional area. (See Figure 3-4.)

+ A positive dc voltage.

3.3.2 4MBARRAY Banks

The 4MBARRAY Banks consist of four almost identical banks. The differences between the banks are minor and involve only two signals: MCL COLD START and CLR REF REQ. The differences are described in Section 3.3.2.7.

Only one bank (bank 0) is described. The description applies equally to the other three banks. Figure 3-9 is a block diagram of array bank 0.

3.3.2.1 Array Bank Components -- The major components of array bank 0 are:

- The array DRAMs
- An 8409 DRAM controller
- Two bi-directional I/O latches
- A bank manager
- A shift register

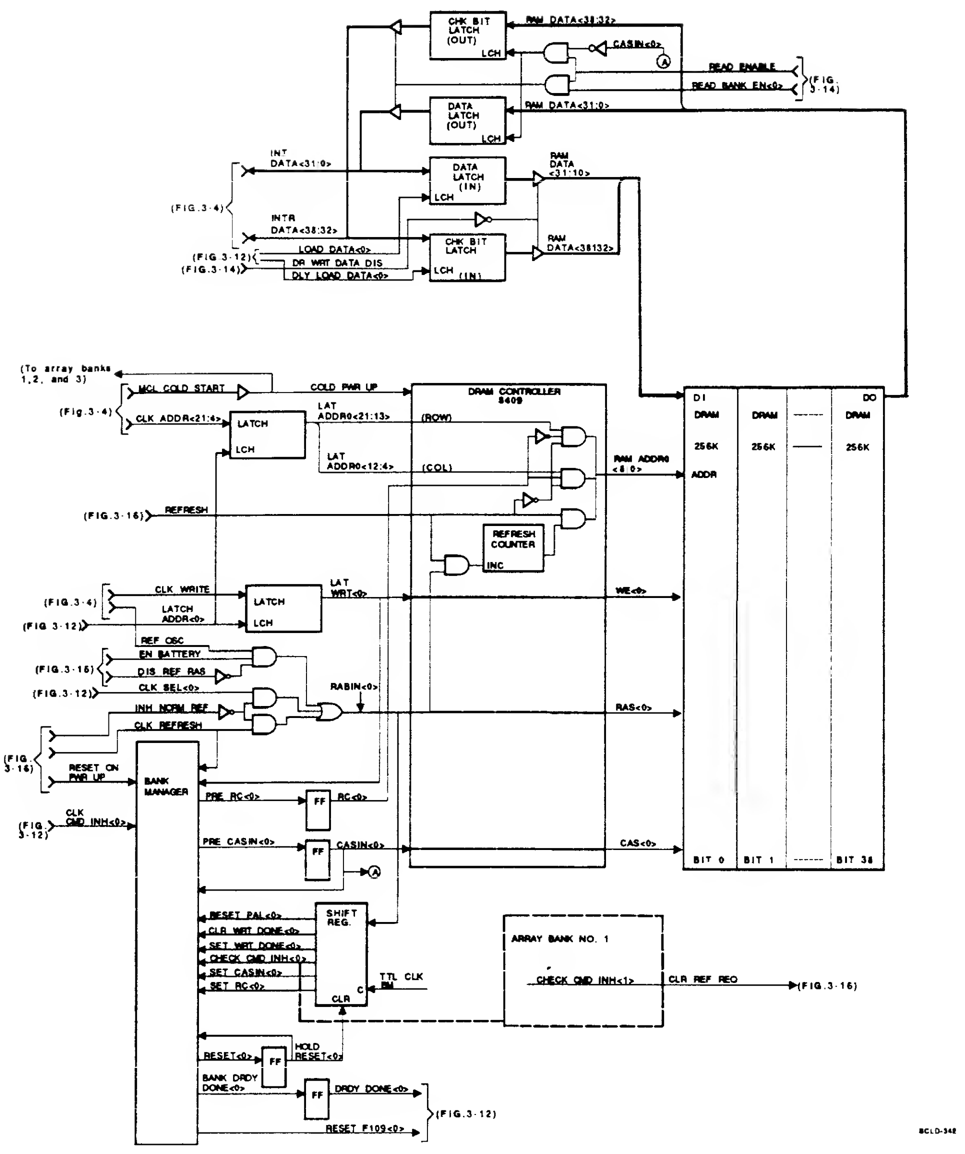


Figure 3-9 4MBARRAY Bank Block Diagram (Bank 0 Shown)

A. Array DRAMs

There are thirty-nine DRAM chips in the array bank. Each DRAM chip has an effective 512 x 512 matrix array providing 256K (262,144) one-bit locations*. The bank provides 256K 39-bit locations or one megabyte of data storage (a 39-bit location contains a 32-bit data longword and 7 ECC check bits).

* One of the row address bits is muxed to the column address within the DRAM to provide ten bits of column addressing. This makes the actual matrix 256 rows by 1024 columns (which still provides 256K bits of storage).

Each bank of DRAMs has a 32-bit input port to receive the write data and a 32-bit output port to supply the read data. It also has a seven-bit input port and a seven-bit output port for the check bits.

The DRAMs receive:

- A nine-bit address (RAM ADDR0<8:0>) which is internally muxed for row and column addressing.
- A write enable signal (WE<0>) to place the DRAMs into a read or write mode.
- A row address strobe (RAS<0>) to strobe in the row address from the address lines.
- A column address strobe (CAS<0>) to strobe in the column address from the address lines.

B. 8409 DRAM Controller

The 8409 DRAM controller supplies the row and column address strobes, the write enable mode signal, and the nine-bit address to the DRAMs. The controller functions to mux the row and column addresses onto the DRAM address lines. During a refresh, it supplies the refresh address to the DRAMs.

C. Bi-Directional I/O Latches

Two bi-directional I/O latches control the data flow from the bank I/O ports to the DRAMs. One latch transfers the data longword (INT DATA<31:0>) while the other transfers the associated check bits (INTR DATA<38:32>).

D. Bank Manager

The bank manager is a PAL state machine used to control the sequencing of the array operations. It generates five signals as listed below.

- PRE RC<0> -- muxes the DRAM address lines in the 8409 controller from row address to column address.
- PRE CASIN<0> -- the column address strobe.
- BANK DRDY DONE<0> -- the data-ready-done signal for the MCL.
- RESET F109<0> -- resets the input parser and indicates that bank 0 is not busy.
- RESET<0> -- resets the bank for another operation.

E. Shift Register

The shift register has six output states that are stepped through for each operation. The six output states are supplied to the bank manager for control of the array sequencing. Specific array functions are performed in each of the register states.

3.3.2.2 Data Flow -- The bank has two data I/O ports. One is for the data longword (INT DATA<31:0>) while the other is for the associated check bits (INTR DATA<38:32>). Each port connects to a bi-directional latch which transfers the data into and out of the DRAMs. The input and output sections of the latches are shown separately in the block diagram to illustrate their functioning.

In a write operation, the write data is applied to the two I/O data ports. The data longword is placed on the INT DATA<31:0> port while the associated check bits are placed on the INTR DATA<38:32> port. With bank 0 selected for the write operation, the input parser will assert LOAD DATA<0> followed by DLY LOAD DATA<0>. LOAD DATA<0> latches the data longword into the "in" section of the data latch while DLY LOAD DATA<0> latches the check bits into the "in" section of the check bit latch. DR WRT DATA DIS from the data-output control is false thereby coupling the longword and check bits from the "in" sections of the latches into the DRAMs as RAM DATA<31:0> and RAM DATA<38:32> respectively.

In a read operation, the read data is coupled from the DRAMs to the data and check bit latches. The longword is applied to the "out" section of the data latch as RAM DATA<31:0> while the check bits are applied to the "out" section of the check bit latch as RAM DATA<38:32>. READ ENABLE from the data-output control is true. When CASIN<0> negates in the strobing sequence (after the arrays have been read), it latches the read data in both latches. With bank 0 selected for the read operation, the data-output control asserts READ BANK EN<0> which transfers the read data from the "out" sections of the latches to the I/O ports.

NOTE

In an octaword read, all four banks are read simultaneously and the data and check bits are loaded into their respective latches. The latched data is not harmed by a refresh that may occur after the arrays are read. In fact, the MCL can take the read data from the latches while the refresh is executing.

3.3.2.3 Array Command Sequencing -- Figure 3-10 is a flow diagram of the array sequencing for command operations (writes and reads). Use the flow diagram along with the block diagram in Figure 3-9 in the following discussion.

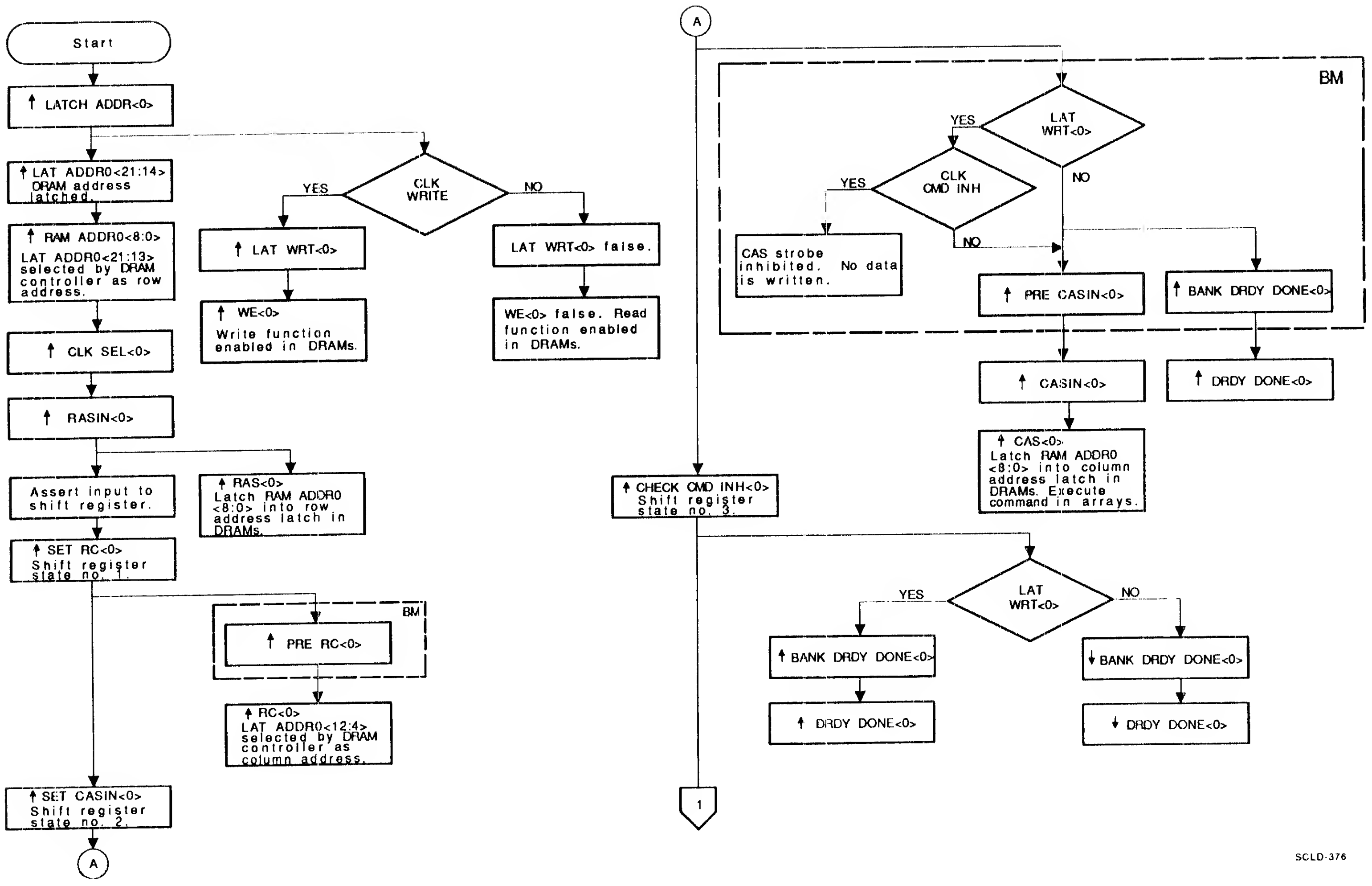

Start
↑ LATCH ADDR<0>
↑ LAT ADDR0<21:14> DRAM address latched.
↑ RAM ADDR0<8:0> LAT ADDR0<21:13> selected by DRAM controller as row address.
↑ CLK SEL<0>
↑ RASIN<0>
Assert input to shift register.
↑ SET RC<0> Shift register state no. 1.
↑ SET CASIN<0> Shift register state no. 2.
A
CLK WRITE
YES
NO
↑ LAT WRT<0>
↑ WE<0> Write function enabled in DRAMs.
LAT WRT<0> false.
WE<0> false. Read function enabled in DRAMs.
↑ RAS<0> Latch RAM ADDR0 <8:0> into row address latch in DRAMs.
BM
↑ PRE RC<0>
↑ RC<0> LAT ADDR0<12:4> selected by DRAM controller as column address.
A
BM
LAT WRT<0>
YES
NO
CLK CMD INH
YES
NO
CAS strobe inhibited. No data is written.
↑ PRE CASIN<0>
↑ BANK DRDY DONE<0>
↑ CASIN<0>
↑ DRDY DONE<0>
↑ CAS<0> Latch RAM ADDR0 <8:0> into column address latch in DRAMs. Execute command in arrays.
↑ CHECK CMD INH<0> Shift register state no. 3.
LAT WRT<0>
YES
NO
↑ BANK DRDY DONE<0>
↑ DRDY DONE<0>
↓ BANK DRDY DONE<0>
↓ DRDY DONE<0>
1
SCLD-376

Figure 3-10 Array Command Flow Diagram (Sheet 1 of 2)

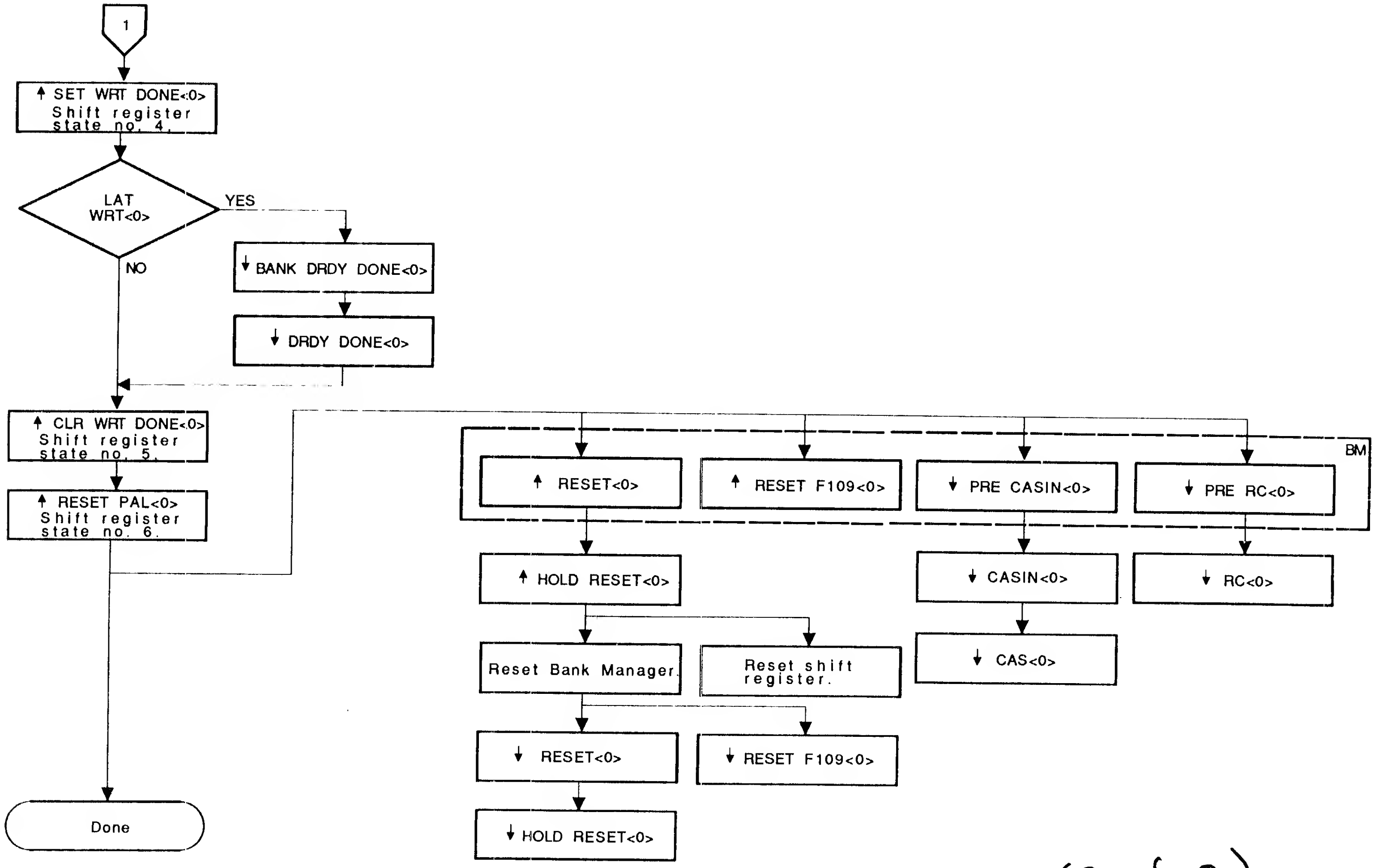


Fig. 3-10 (2 of 2)

SCLD-377

Figure 3-10 Array Command Flow Diagram (Sheet 2 of 2)

An asserted LATCH ADDR<0> is received from the input parser to latch up the array address (CLK ADDR<21:4>*) and the commanded function (CLK WRITE).

If CLK WRITE is true (write operation), LAT WRT<0> is output from the latch to the DRAM controller which in turn asserts WE<0> to the DRAMs. WE<0> places the DRAMs into the write mode of operation. If CLK WRITE is false, LAT WRT<0> and WE<0> are also false and the DRAMS are placed into the read mode of operation.

The output from the address latch (LAT ADDR0<21:4>) is divided into two nine bit sections; LAT ADDR0<21:13> and LAT ADDR0<12:4>. LAT ADDR0<21:13> is the array row address and is applied to the row input of the DRAM controller. LAT ADDR0<12:4> is the array column address and is applied to the column input of the DRAM controller. Initially, the controller selects the row address for transfer to the DRAMS as the RAM ADDR0<8:0> address input.

CLK SEL<0> is received from the input parser to start the array strobing sequence. CLK SEL<0> asserts RASIN<0> to the DRAM controller which in turn asserts RAS<0> to the DRAMs. RAS<0> latches the row address now on the DRAM address lines, into the row address latch in the DRAMs.

RASIN<0> also provides the input needed to start the shift register. On the next TTL CLK BM clock, RASIN<0> is clocked into the shift register which attains state number 1 and asserts SET RC<0> to the bank manager. The bank manager asserts PRE RC<0> causing RC<0> to assert to the DRAM controller. RC<0> switches the address mux in the controller so that the column address LAT ADDR0<12:4> is muxed onto the DRAM address lines as RAM ADDR0<8:0>.

On the next TTL CLK BM clock, the shift register advances to state number 2 and asserts SET CASIN<0> to the bank manager. Upon receiving SET CASIN<0>, the bank manager checks the state of LAT WRT<0>. If LAT WRT<0> is true (write operation), the manager checks the CLK CMD INH input from the input parser to see if the write operation should execute. If CLK CMD INH is false, the manager asserts PRE CASIN<0> which becomes CASIN<0>. CASIN<0> is applied to the DRAM controller which asserts CAS<0> to the DRAMS. CAS<0> latches the column address now on the DRAM address lines, into the column address latch in the DRAMs. The data at the DRAM's input port is then written into the arrays. If CLK CMD INH is true, the column address strobe is not generated and no data is written into the arrays.

* Address bits CLK ADDR<25:22> are not used by the MAR4 array banks.

If LAT WRT<0> is false (read operation), the bank manager asserts PRE CASIN<0> which generates CAS<0> to latch up the column address in the arrays. The arrays are then read and the read data is available at the data ports as INT DATA<31:0> and INTR DATA<38:32>. In addition, the bank manager asserts BANK DRDY DONE<0> which in turn asserts DRDY DONE<0> to the input parser.

DRDY DONE<0> is asserted for both a read and a write operation, however, it is asserted earlier in the sequence for a read operation. This is because in a read operation, the MCL must come back to select the MAR4 for a read and take the read data. BMAR SEND NO CMD inhibits the MCL from issuing any commands to the MAR4 until this occurs. In a write operation BMAR SEND NO CMD is not asserted and the write operation must be completed when DRDY DONE<0> is asserted as the MCL may immediately command another write operation to the same bank.

The next clock moves the shift register to state number 3. In state number 3 the shift register asserts CHECK CMD INH<0> to the bank manager. If this is a write operation, the assertion of CHECK CMD INH<0> causes BANK DRDY DONE<0> and DRDY DONE<0> to be asserted to the input parser. If this is a read operation, the assertion of CHECK CMD INH<0> causes BANK DRDY DONE<0> and DRDY DONE<0> to negate (they were asserted during the preceeding state of the shift register).

The next clock moves the shift register to state number 4 which asserts SET WRT DONE<0> to the bank manager. If this is a write operation, BANK DRDY DONE<0> and DRDY DONE<0> are negated (they were asserted during the preceeding state of the shift register). If this is a read operation, no action occurs and the shift register moves to state number 5.

In state number 5 the shift register asserts CLR WRT DONE<0>. CLR WRT DONE<0> is used in a refresh operation (Section 3.3.2.4). No action occurs for normal commands.

The next clock moves the shift register to its final state, state number 6, where the register asserts RESET PAL<0> to the bank manager. In response to RESET PAL<0>, the bank manager:

- Negates its PRE RC<0> output (followed by the negation of RC<0>).
- Negates its PRE CASIN<0> output (followed by the negation of CASIN<0> and CAS<0>).
- Asserts RESET F109<0> to the input parser to reset the parser logic and indicate that bank 0 is no longer busy.
- Asserts RESET<0> to reset the bank in preparation for another operation.

RESET<0> asserts HOLD RESET<0> which clears the shift register, and is fed back into the bank manager to reset it. In response to HOLD RESET<0>, the manager negates RESET F109<0> and RESET<0>. The negation of RESET<0> causes HOLD RESET<0> to negate.

The array bank is now in its initialized state and is prepared to initiate another command operation or a refresh.

3.3.2.4 Array Refresh Sequencing --

NOTE

Refresh sequencing begins in the refresh logic (Section 3.3.6.1). Hence the refresh logic describes the signal sequence that precedes the array refresh sequence described here.

Figure 3-11 is a flow diagram of the array sequencing during a refresh operation. Refer to it and the block diagram of Figure 3-9 during the following discussion.

A refresh sequence uses the shift register and bank manager much like a command sequence except that many of the command signals and functions are not used.

When a refresh is to occur, REFRESH is asserted from the refresh logic followed by CLK REFRESH. REFRESH and CLK REFRESH are applied to all four banks which are refreshed simultaneously.

REFRESH is applied to the 8409 DRAM controller where it disables the row and column address inputs and substitutes an address from a refresh counter. The counter output addresses the row in the array that is to be refreshed. REFRESH also enables the counter to be incremented by RASIN<0>.

CLK REFRESH is applied to the bank manager where it inhibits four of the five bank manager functions. The functions inhibited are listed below.

- PRE RC<0> -- refreshes are done a row at a time. No column address is needed.

- PRE CASIN<0> -- due to the above, no column address strobe is needed.

- BANK DRDY DONE<0> -- the state of BMAR SEND NO CMD informs the MCL when the refresh is completed.

- RESET F109<0> -- the input parser was not used so does not need to be reset, and the refresh logic knows when it has completed the refresh.

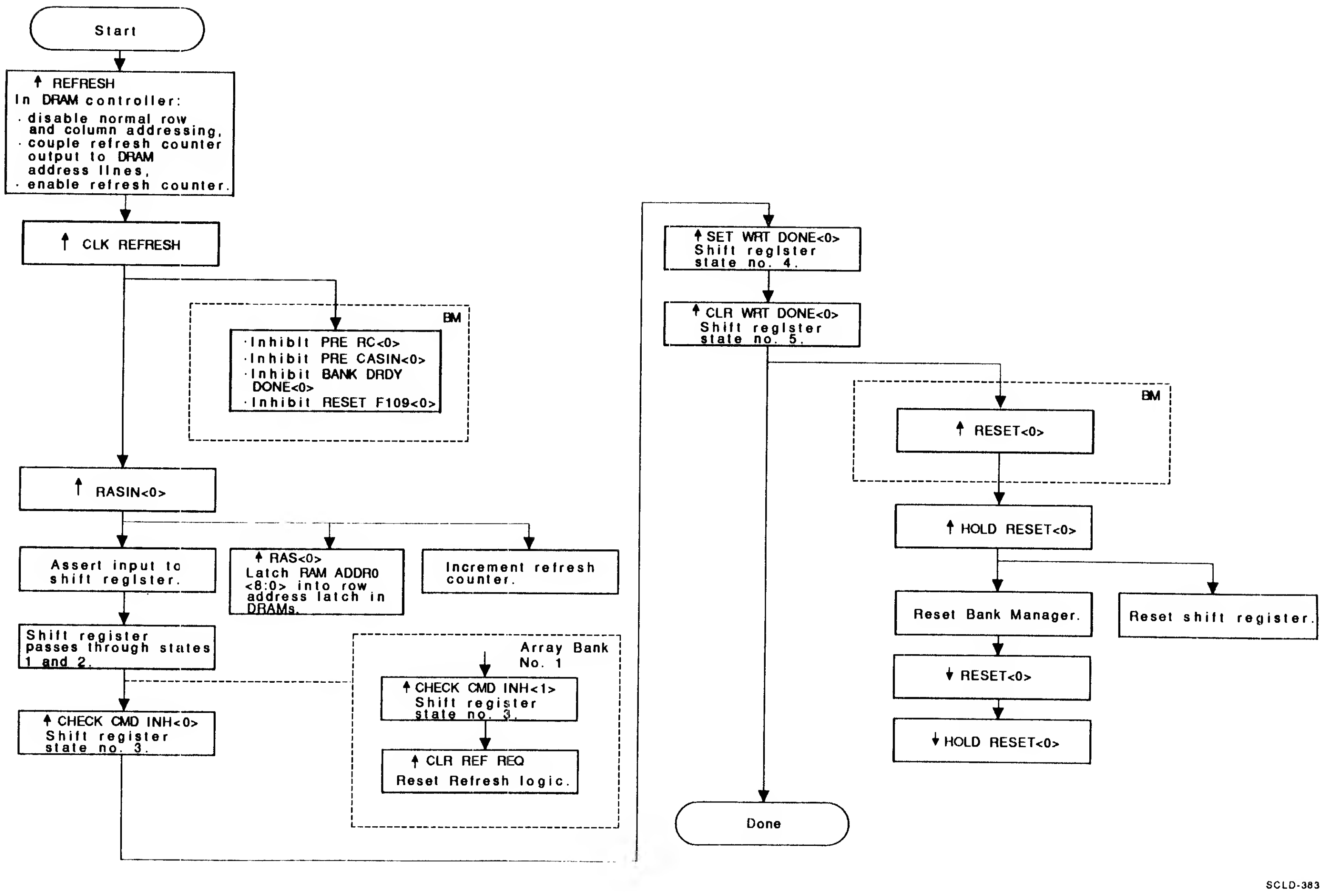

Start
↑ REFRESH
In DRAM controller:
· disable normal row and column addressing,
· couple refresh counter output to DRAM address lines,
· enable refresh counter.
↑ CLK REFRESH
BM
·Inhibit PRE RC<0>
·Inhibit PRE CASIN<0>
·Inhibit BANK DRDY DONE<0>
·Inhibit RESET F109<0>
↑ RASIN<0>
Assert input to shift register.
↑ RAS<0>
Latch RAM ADDR0 <8:0> into row address latch in DRAMs.
Increment refresh counter.
Shift register passes through states 1 and 2.
Array Bank No. 1
↑ CHECK CMD INH<1>
Shift register state no. 3.
↑ CLR REF REQ
Reset Refresh logic.
↑ CHECK CMD INH<0>
Shift register state no. 3.
↑ SET WRT DONE<0>
Shift register state no. 4.
↑ CLR WRT DONE<0>
Shift register state no. 5.
BM
↑ RESET<0>
↑ HOLD RESET<0>
Reset Bank Manager.
Reset shift register.
↓ RESET<0>
↓ HOLD RESET<0>
Done
SCLD-383

Figure 3-11 Array Refresh Flow Diagram

CLK REFRESH also asserts RASIN<0> which in turn:

- Increments the refresh counter to the address of the row that is to be refreshed.
- Asserts RAS<0> to the DRAMs to latch the row address from the refresh counter into the row address latch in the DRAMs.
- Provides an input to the shift register for state sequencing.

The shift register is clocked from state number 1 through state number 4 as the arrays are being refreshed.

When the shift register enters state number 5, it asserts CLR WRT DONE<0> to the bank manager to reset the array bank. In response to CLR WRT DONE<0>, the manager asserts RESET<0> causing HOLD RESET<0> to assert and clear the shift register. HOLD RESET<0> is also fed back into the bank manager causing it to negate RESET<0> which then negates HOLD RESET<0>. The array bank is now initialized for a command operation or another refresh.

When the shift register in bank 1 reaches state number 3, the CHECK CMD INH<1> output from the register changes to CLR REF REQ and is applied to the refresh logic. CLR REF REQ clears the refresh request and prepares the refresh logic for another refresh cycle.

3.3.2.5 Battery Mode --

NOTE

Battery mode events in the refresh logic precede those in the array banks. Section 3.3.6.2 describes battery mode operation of the refresh logic.

Battery mode is automatically entered when there is an interruption in system power. While in battery mode, the object is to save the data in the arrays until power is resumed. Consequently, only refresh cycles are allowed -- command operations (writes and reads) are inhibited. To reduce power drain on the battery, normal refresh sequences are suspended and battery mode refreshes are executed. Battery mode refreshes use only the barest minimum of logic thereby making lower power demands on the battery.

Battery operation cannot retain the data in the arrays for long-term power outages. The array data is retained for only ten minutes. After this the data is no longer reliable.

When battery mode is enabled, the array banks receive the refresh oscillator clock (REF OSC) from the MCL, and five signals from the refresh logic. The signals received in battery mode are described below.

- REF OSC -- from MCL -- initiates a refresh cycle every 14.9 microseconds.
- REFRESH -- from refresh logic -- REFRESH functions as during a normal refresh operation. It causes the address mux in the 8409 DRAM controller to inhibit the row and column address from the DRAM address lines and enables the output of the refresh counter onto the address lines. It also enables the refresh counter to be incremented by RASIN<0>.
- RESET ON PWR UP -- from refresh logic -- RESET ON PWR UP is applied to the bank manager where it inhibits all manager functions except RESET<0>. RESET<0> is asserted causing HOLD RESET<0> to assert which in turn holds the shift register reset.
- INH NORM REF -- from refresh logic -- INH NORM REF disables the portion of the RASIN<0> logic used during command and normal refresh operations.
- EN BATTERY -- from refresh logic -- EN BATTERY enables the portion of the RASIN<0> logic that is used for battery operation.
- DIS REF RAS -- from refresh logic -- DIS REF RAS negates RASIN<0> after each refresh to conserve power.

After RESET ON PWR UP has disabled the bank manager and the shift register, and REFRESH has set up the refresh counter to supply the DRAM's address, REF OSC starts clocking refresh cycles. REF OSC is applied to the RASIN<0> logic enabled by EN BATTERY and causes RASIN<0> to assert. RASIN<0> increments the refresh counter to the address of the row to be refreshed. It also causes RAS<0> to assert to the DRAMs where it strobes in the row address on the DRAM's address lines.

A row refresh takes about 150 ns. REF OSC holds the RASIN<0> logic enabled for 7,300 ns (see Figure 3-19). To shorten the duration of RASIN<0>, DIS REF RAS is asserted and negates RASIN<0> 250 ns after the leading edge of REF OSC. Thus RASIN<0> asserts for only 250 ns which is more than enough time for the array row to be refreshed. Reducing the duration of RASIN<0> significantly reduces the refresh current in the arrays thereby reducing the drain on the battery power.

When system power returns, the battery mode signals are removed and normal operation is resumed.

3.3.2.6 **Cold Start** -- MCL COLD START is asserted to the array banks from the MCL when system power is applied after a long power interruption (greater than ten minutes). The purpose of MCL COLD START is to inhibit the arrays from operating until the system has powered up and initialized. The current drawn by the arrays would have a detrimental effect on the system power-up process.

MCL COLD START is applied to array bank 0 where it becomes COLD PWR UP. COLD PWR UP is applied to the DRAM controller and also sent to the DRAM controllers in the other three banks.

COLD PWR UP functions to tri-state the DRAM controller outputs which effectively removes the signal inputs to the DRAMs.

In order for the DRAM controller to attain its tri-state condition, it must be in the command mode of operation (not refresh). Consequently, MCL COLD START is also applied to the refresh logic where it holds REFRESH negated.

3.3.2.7 **Array Bank Differences** -- There are two minor areas where the four array banks are not exactly the same. These areas involve the MCL COLD START and CHECK CMD INH<1> signals.

MCL COLD START from the MCL is applied only to array bank 0. In array bank 0, MCL COLD START becomes COLD PWR UP. COLD PWR UP is applied to the DRAM controller in bank 0 and to the DRAM controllers in the other three banks.

The CHECK CMD INH<1> output from the shift register in bank 1 (and only bank 1) is renamed CLR REF REQ and applied to the refresh logic.

In all other respects, the four array banks are exactly identical.

3.3.3 Input Parser

The input parser performs the following functions:

- Performs a parity check on the command/address data received from the MCL.
- Generates the write inhibit signal when a write operation is to be aborted.
- Generates a "data ready done" signal for the MCL when the commanded operation has been completed.
- Establishes that the MAR4 board has been selected by the MCL.

- Decodes the input command from the MCL and generates the enabling signals for the array bank(s).

- Generates an "array not busy" signal for the refresh logic when the array banks are not busy.

The functions are illustrated in Figure 3-12 and described below.

3.3.3.1 **Command/Address Parity Check** -- The four-bit command (CLK WRITE, CLK OCTA, CLK STADDR<1:0>) and the array address (CLK ADDR<25:4>) are applied to a parity checker along with the command/address parity bit (CLK CMD ADDR PARITY). If the checker detects a parity error in the command/address, it asserts BMAR CMD ADDR PAR ERR and INT CMD ADDR PAR ERR.

BMAR CMD ADDR PAR ERR is returned to the MCL to flag the fact that a command/address parity error has occurred. INT CMD ADDR PAR ERR is used in the "write inhibit" function.

3.3.3.2 **Write Inhibit** -- INT CMD ADDR PAR ERR causes CLK CMD INH to assert to the array banks where it disables the array write function. If a write operation is being executed, no data will be written into the arrays. If a read operation is being executed, the operation will execute normally.

CLK CMD INH is also asserted by WRITE INHIBIT from the ECC/DPARITY logic. WRITE INHIBIT is derived from the write inhibit bit received as part of the write data from the MCL. If a parity error was detected on the write data in the ECC/DPARITY logic, the logic asserts DATA PAR ERR to the input parser and prevents WRITE INHIBIT from asserting CLK CMD INH. This is done because the write inhibit bit could be causing the parity error*, thus it is uncertain if the data was meant to be written. Under these conditions the data is written as a precaution.

3.3.3.3 **Data Ready Done** -- The data-ready-done signals received from the array banks are ORed to generate BRD DRDY DONE which in turn asserts BMAR DATA RDY DONE. BMAR DATA RDY DONE is sent to the MCL to indicate that the commanded operation has been executed.

In a write operation, the OR gate will receive DRDY DONE<X> from the bank that was written. BMAR DATA RDY DONE indicates to the MCL that it can send another command (read or write) to the same bank.

* The write inhibit bit is included in the write data parity check.

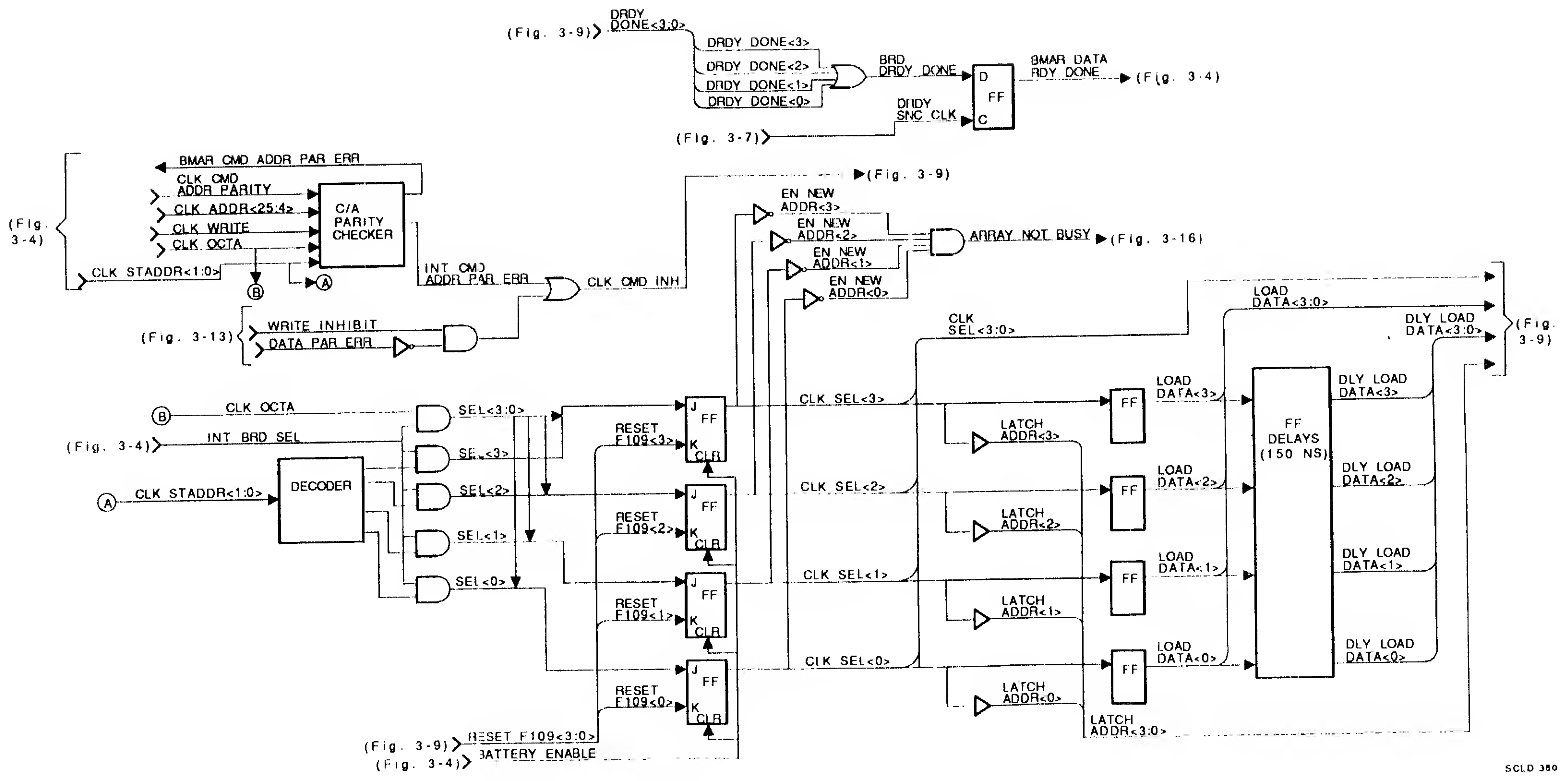


Figure 3-12 Input Parser Block Diagram

For a longword read operation, the OR gate will receive DRDY DONE<X> from the bank that was read. For an octaword read operation, the OR gate will receive four data-ready-done signals at the same time because the four banks are read simultaneously. In both cases BMAR DATA RDY DONE indicates to the MCL that the array(s) has (have) been read and the read data is available to be taken.

3.3.3.4 MAR4 Board Selection -- INT BRD SEL is asserted by the MCL when the MAR4 board has been selected for the commanded operation. INT BRD SEL enables five AND functions from which are derived all of the array enabling signals developed in the input parser.

3.3.3.5 Generation of Array Bank Signals -- Three of the four command bits (CLK OCTA and CLK STADDR<1:0>) are used in the input parser for bank selection and control. The fourth bit (CLK WRITE) is used in the input parser only for command/address parity checking. CLK WRITE is sent directly to the array banks where it establishes either the write or read mode of operation (see Figure 3-4).

When a longword write or read is being executed, starting address bits CLK STADDR<1:0> are decoded to enable one of four AND functions. The enabled AND function asserts SEL<X> where X is the bank selected by the starting address. SEL<X> sets a J-K flip-flop which in turn asserts CLK SEL<X> and then LATCH ADDR<X>. CLK SEL<X> is applied to array bank X where it initiates the RAS and CAS strobing sequence. LATCH ADDR<X> is applied to bank X where it latches up the array address and the CLK WRITE command bit.

One cycle later LOAD DATA<X> asserts to the selected array bank to load the write data longword into the "in" section of the bi-directional data latch. Approximately 135 ns later DLY LOAD DATA<X> asserts to load the associated ECC check bits into the "in" section of the bi-directional check bit latch. The 135 ns delay allows time for the check bits to be generated by the ECC/DPARITY Logic.

Note that the bank signals developed by the input parser are independent of whether the operation is a read or a write. CLK SEL<X> and LATCH ADDR<X> perform their functions for both read and write operations. LOAD DATA<X> and DLY LOAD DATA<X> are used only for a write operation.

RESET F109<X> is received from the selected bank after the commanded operation has been executed. RESET F109<X> resets the J-K flip-flop resulting in the negation of CLK SEL<X>, LATCH ADDR<X>, LOAD DATA<X>, and DLY LOAD DATA<X>.

When an octaword read is being executed, CLK OCTA is asserted which in turn asserts all four of the SEL<3:0> inputs to the J-K flip-flops. This sets all four flip-flops causing CLK SEL<3:0> and LATCH ADDR<3:0> to assert to all the array banks. CLK SEL<3:0> starts the RAS and CAS strobing sequence in the banks while LATCH ADDR<3:0> latches up the array address and CLK WRITE function. After the arrays have been read, RESET F109<3:0> is received from the banks to reset the four J-K flip-flops and negate the bank signals developed by the input parser.

3.3.3.6 **Array Not Busy** -- An AND function senses the state of the J-K flip-flops and asserts ARRAY NOT BUSY when they are all reset. If any command is executing, at least one of the flip-flops will be set causing ARRAY NOT BUSY to be false. ARRAY NOT BUSY is sent to the refresh logic to indicate that the arrays are free and a refresh operation may occur.

3.3.3.7 **Battery Mode** -- In battery mode, BATTERY ENABLE is received from the MCL to hold the J-K flip-flops in the reset state. Therefore, when power returns and the system comes out of battery mode, the flip-flops will be in their initialized (reset) state. If any of the flip-flops were to power-up to the set state, false signals would be asserted to the array banks.

3.3.4 ECC/DPARITY

The ECC/DPARITY logic performs the following functions during a write operation:

- Generates partial check bits on the write data longword.
- Provides WRT INHIBIT to the input parser to abort a write operation.
- Performs a parity check on the write data.
- Generates an INT BAD DATA bit for the final ECC logic.

The functions are illustrated in Figure 3-13 and described in the following sections.

3.3.4.1 **ECC Check Bits** -- The data longword (INT DATA<31:0>) is received from the write-data flip-flops (Figure 3-4) and applied to an ECC generator. The generator develops seven ECC partial check bits unique to the data longword. The check bits are output as PCB<6:0> to final ECC logic on the MAR4 board.

3.3.4.2 **Write Inhibit** -- CLK WRT INHIBIT is received from the MCL and becomes CLK2 WRT INHIBIT. CLK2 WRT INHIBIT is output as WRT INHIBIT to the input parser. WRT INHIBIT is asserted when the data longword is not to be written into the array.

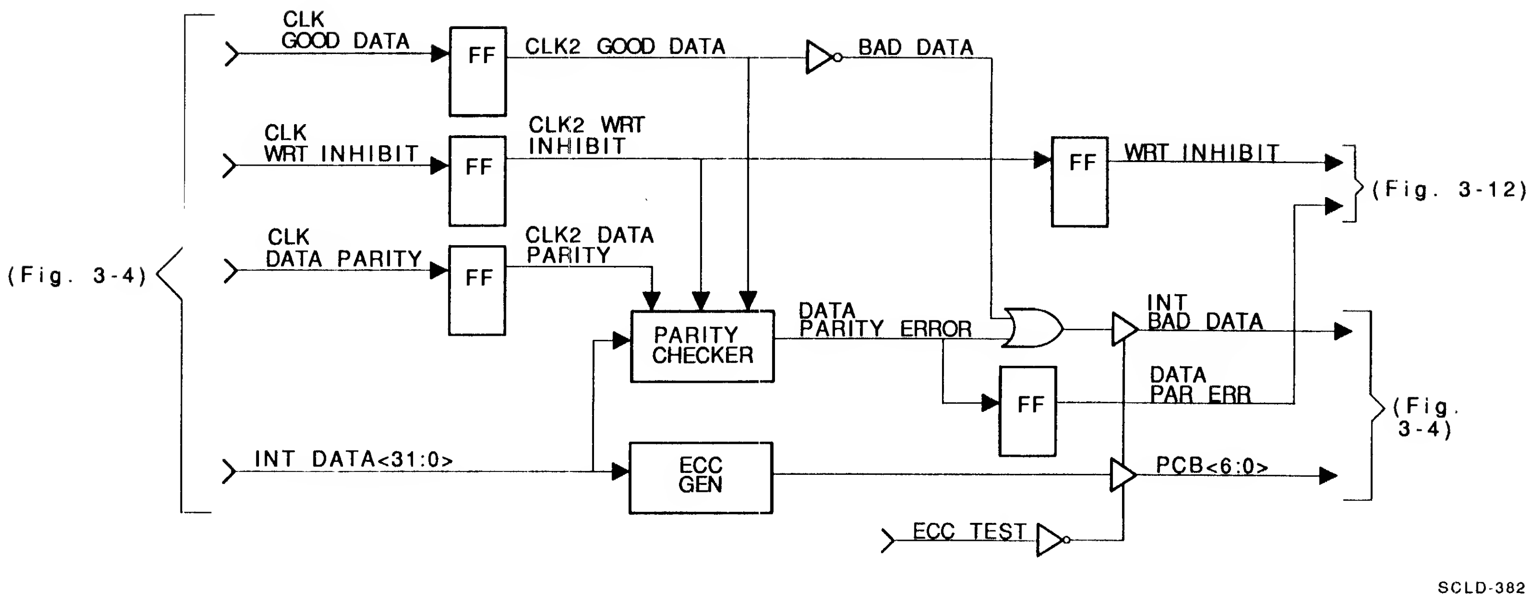


Figure 3-13 ECC/DPARITY Block Diagram

3.3.4.3 Write Data Parity Check -- The data longword (INT DATA<31:0>) is applied to a parity checker along with CLK2 WRT INHIBIT, CLK2 GOOD DATA, and CLK2 DATA PARITY. CLK2 GOOD DATA and CLK2 DATA PARITY are respectively derived from CLK GOOD DATA and CLK DATA PARITY received from the MCL (Figure 3-4). CLK2 DATA PARITY is the parity bit for the data longword and for the CLK2 GOOD DATA and CLK2 WRT INHIBIT bits. If a parity error is detected, DATA PARITY ERROR is asserted which in turn asserts INT BAD DATA to the final ECC logic on the MAR4.

The detection of a data parity error also asserts DATA PAR ERR to the input parser where it disables the write inhibit function. This is done because the write inhibit bit is included in the parity check and could be the bit causing the parity error. Because it is uncertain whether the data is meant to be written, the write operation is allowed to execute even though WRT INHIBIT is true.

3.3.4.4 INT BAD DATA -- If the write data longword contains bad data, CLK2 GOOD DATA will be false causing BAD DATA to assert. The true state of BAD DATA will assert INT BAD DATA (if not already asserted by a data parity error) to the final ECC logic on the MAR4. Here the INT BAD DATA bit is combined with the partial check bits to produce the final check bits (INTR DATA<38:32>). An analysis of the final ECC bits will show if the INT BAD DATA bit is true, but it will not show if it was asserted by a parity error or by an asserted bad-data bit.

3.3.5 Data Output Control

The data-output control functions to:

- Enable the MAR4 for a read.
- Select the bank for the read data transfer to the MCL.
- Enable the MAR4 board for the data transfer.
- Control the transfer of read data.
- Generate the DRDY SNC CLK clock.

The functions are illustrated in Figure 3-14 and described in the following sections.

3.3.5.1 MAR4 Read Enable -- Board select signal INT BRD SEL is applied to the data-output control and asserts BRD SEL. The read function signal (negated CLK WRITE) is applied to the data-output control and asserts READ. READ and BRD SEL are ANDed and applied to an OR function which asserts READ ENABLE and DR WRT DATA DIS.

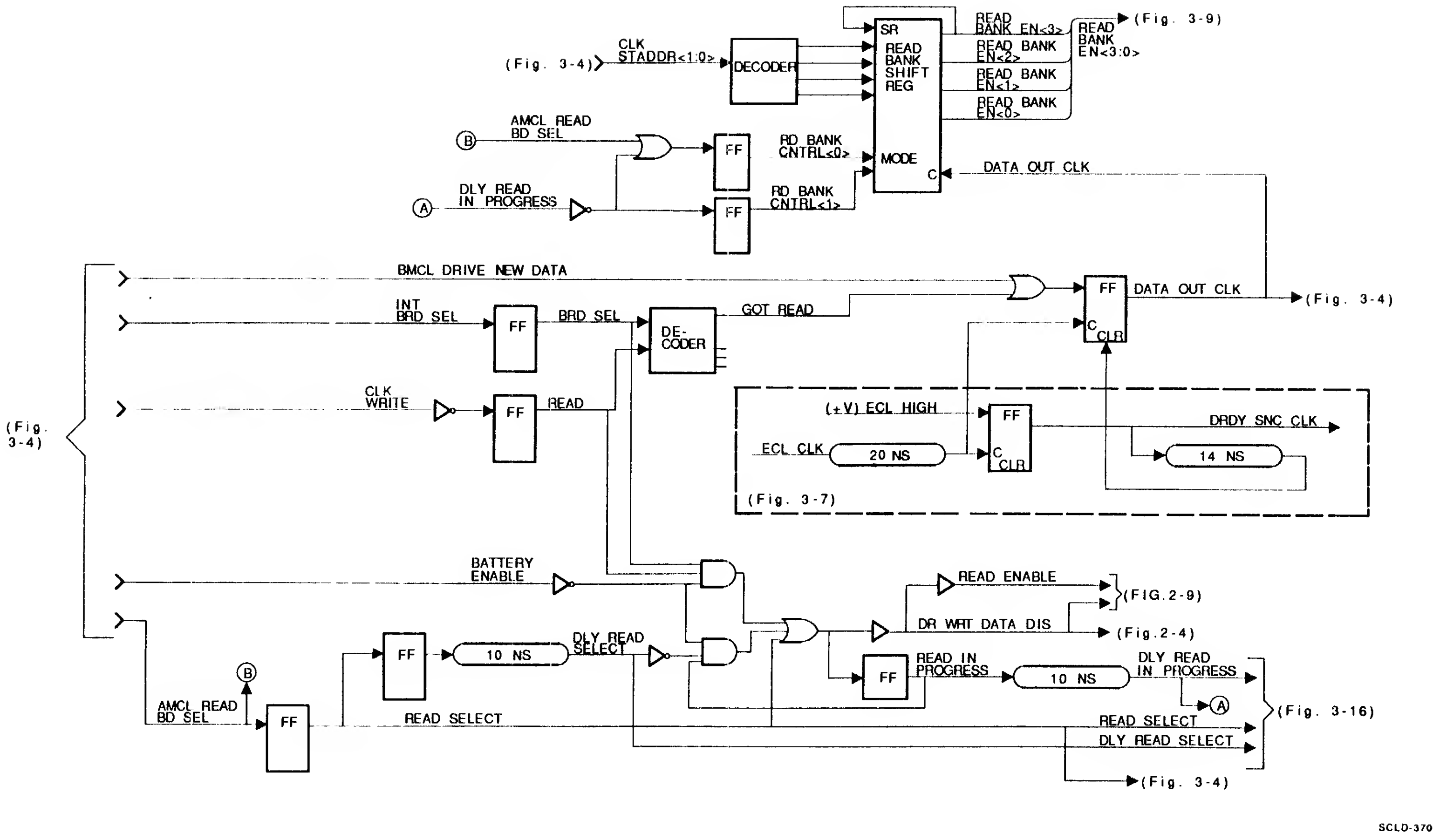


Figure 3-14 Data Output Control Block Diagram

READ ENABLE is applied to the four array banks where it enables the "out" sections of the bi-directional latches. DR WRT DATA DIS is applied to the array banks where it disables the "in" sections of the bi-directional latches.

DR WRT DATA DIS is also applied to the write-data flip-flops where it disables the data-in path from the NAB to the MAR4's internal data bus.

The OR function further prepares the MAR4 for a read by asserting READ IN PROGRESS which then asserts DLY READ IN PROGRESS. DLY READ IN PROGRESS is sent to the refresh logic where it asserts BMAR SEND NO CMD to the MCL.

READ IN PROGRESS is fed back to the OR function through a latch to hold READ ENABLE, DR WRT DATA DIS, and DLY READ IN PROGRESS asserted. This holds the MAR4 board in the read enabled state until the MCL takes the read data.

3.3.5.2 Bank Select -- The asserted BRD SEL and READ signals are applied to a decoder causing it to output GOT READ. GOT READ causes the assertion of DATA OUT CLK which is applied to the clock input of a read-bank shift register. The shift register is presently in the "load" mode; hence, DATA OUT CLK functions to load the shift register with the bank starting address. The bank starting address is obtained from a decoder which uses the CLK STADDR<1:0> starting address to assert one of four inputs to the shift register.

The mode of the shift register is controlled by RD BANK CNTRL<1:0>. RD BANK CNTRL<1:0> is derived from DLY READ IN PROGRESS and AMCL READ BD SEL. The shift register modes and the mode control signals are shown in Table 3-3.

Table 3-3 Read Bank Shift Register Modes

DLY READ IN PROGRESS	AMCL READ BD SEL	RD BANK CNTRL<1>	RD BANK CNTRL<0>	Mode
0	0	1	1	Load
0	1	1	1	Load
1	0	0	0	No op
1	1	0	1	Shift right

1 = asserted state
2 = negated state

When DATA OUT CLK clocks the register, DLY READ IN PROGRESS is not yet asserted. Therefore RD BANK CNTRL<1:0> is 1:1 and the register is in the load mode.

The shift register outputs READ BANK EN<X> to the selected bank where it couples the read data from the "out" sections of the bi-directional latches to the internal data bus.

When DLY READ IN PROGRESS asserts, RD BANK CNTRL<1:0> becomes 0:0 and the register attains a no-op state.

3.3.5.3 **MAR4 Data Transfer Enable** -- When the MCL is ready to take the read data from the bi-directional latches in the array bank(s), it asserts AMCL READ BD SEL to prepare the MAR4 board for the read data transfer.

AMCL READ BD SEL places the read-bank shift register into the "shift" mode of operation.

In addition, AMCL READ BD SEL asserts READ SELECT to the read-data flip-flops where it enables the flip-flop outputs onto the NAB (Figure 3-4).

READ SELECT is also applied to the refresh Logic where it causes the negation of BMAR SEND NO CMD.

Another function of READ SELECT is to take control of the data-output control OR function so that all the OR gate outputs become a function of AMCL READ BD SEL. This is done by applying READ SELECT to the OR gate and then releasing the OR gate latch. The latch is released by DLY READ SELECT which is a delayed form of READ SELECT. READ ENABLE, DR WRT DATA DIS, and DLY READ IN PROGRESS are held asserted by READ SELECT and continue their functions of preparing the MAR4 for a data transfer.

DLY READ SELECT is also sent to the refresh logic where it inhibits BMAR SEND NO CMD. This prevents a "race" condition that may occur at the end of the read operation wherein BMAR SEND NO CMD may assert due to READ SELECT negating before DLY READ IN PROGRESS negates (see Figure 3-16).

3.3.5.4 **Control of Read Data Transfer** -- The MCL asserts BMCL DRIVE NEW DATA to transfer the read data from the MAR4 internal bus to the NAB. BMCL DRIVE NEW DATA asserts DATA OUT CLK which clocks the data on the internal bus into the read-data flip-flops. The flip-flop outputs are already enabled to the NAB by READ SELECT; thus the read data in the flip-flops is transferred to the MCL.

In addition, DATA OUT CLK increments the read-bank shift register (now in "shift" mode) advancing the READ BANK EN<X> output to the next bank. If the command was an octaword read, the read data in the next bank is transferred from the "out" sections of the bi-directional latches in the bank to the MAR4's internal bus to await the next BMCL DRIVE NEW DATA signal from the MCL.

When all the read data has been transferred to the MCL, AMCL READ BD SEL is negated to complete the read data transfer.

3.3.5.5 DRDY SNC CLK -- The DRDY SNC CLK clock is generated from a delayed ECL clock. The delayed ECL CLK sets a flip-flop which is cleared by feedback to produce the DRDY SNC CLK waveform shown in Figure 3-8. Note from the block diagram of Figure 3-14 that when DATA OUT CLK is asserted, it is in phase with and has the same waveshape as DRDY SNC CLK.

DRDY SNC CLK is applied to the input parser where it clocks BMAR CMD ADDR PAR ERR and BMAR DATA RDY DONE. It is also applied to the refresh logic where it clocks BMAR SEND NO CMD.

The DRDY SNC CLK clock is discussed further in Section 3.3.1 under Clock Logic.

3.3.5.6 Battery Mode -- When the system goes into battery mode, BATTERY ENABLE is applied to the data-output control where it disables inputs to the OR function. This prevents false "read enable" signals from being asserted when power returns. During the power-up, BATTERY ENABLE remains true until all the data-output control logic has been initialized.

3.3.6 Refresh

NOTE

The refresh logic initiates and controls the refresh operations in the 4MBARRAY banks. The refresh events occurring in the array banks are a result of the actions described in this section; therefore, the refresh description given in the section on the 4MBARRAY banks (Section 3.3.2) complements the sequences described here.

The array matrices in the MAR4 boards are volatile and require refreshing at least every 4 milliseconds to retain their stored data. All four array banks on a MAR4 are refreshed at the same time. The arrays are refreshed a row at a time with the same row in each bank being refreshed simultaneously.

An array bank has 256 rows*; therefore 256 refresh cycles must be executed within 4 ms. Refreshes are initiated by a refresh

* The array matrices are not square. They are arranged in a rectangular pattern of 256 rows by 1024 columns. (See Section 3.3.2.1[A].)

oscillator signal (REF OSC) obtained from the MCL. REF OSC has a time period of 14.9 microseconds (see Figure 3-15) allowing 268 REF OSC cycles (hence 268 refresh cycles) to occur within a 4 ms time period. This is more than the 256 refresh cycles required; thus the arrays are refreshed well within the 4 ms time limit.

Refreshes occur in two modes of operation: normal and battery. In normal mode each REF OSC cycle initiates a refresh request. The refresh request asserts BMAR SEND NO CMD to inhibit new commands from the MCL, and waits for any current operation to finish. When the arrays are free (ARRAY NOT BUSY true), the refresh cycle executes. After the refresh cycle has executed, BMAR SEND NO CMD is negated and new commands can be issued by the MCL.

As stated in Table 3-1, REF OSC H is used in four MAR4s while the other four receive REF OSC L. This evens out the power drain by executing refreshes in four MAR4s at a time instead of all eight. Even the four MAR4s receiving a given phase of REF OSC may not refresh simultaneously. Any MAR4 that has to wait for a current operation to finish will refresh later than one that was free when it received REF OSC. The only refreshes that are simultaneous are the refreshes in the four banks of a MAR4 board.

Battery mode is used during short periods of power interruption (less than ten minutes) to retain the stored data in the arrays until power resumes. In battery mode, all commands are inhibited and only refreshes are executed. Battery mode utilizes separate logic in the refresh logic and in the 4MBARRAY banks to minimize the power drain on the battery.

3.3.6.1 **Normal Mode** -- Figure 3-16 is a block diagram of the refresh logic. Figure 3-17 is a flow diagram of a normal refresh cycle. Refer to these figures during the following discussion.

The leading edge of REF OSC clocks a flip-flop causing REFRESH REQUEST to assert (MCL DIS REF negated). REFRESH REQUEST asserts SET REF REQ and then REF REQ. SET REF REQ is fed back to clear the input flip-flop thereby causing REFRESH REQUEST and SET REF REQ to negate.

The REF REQ flip-flop is a J-K type; hence, it will stay asserted until the refresh cycle executes and CLR REF REQ is received to enable the K input. The true state of REF REQ indicates a refresh request has been made and will execute as soon as the arrays are free.

SET REF REQ and REF REQ are applied to an OR gate which asserts SET SEND NO CMD and then BMAR SEND NO CMD. BMAR SEND NO CMD inhibits any new commands from the MCL. SET REF REQ asserts before REF REQ asserts; therefore, it is used to assert BMAR SEND NO CMD so that new commands from the MCL are inhibited sooner. REF REQ is needed to hold BMAR SEND NO CMD asserted after SET REF REQ negates.

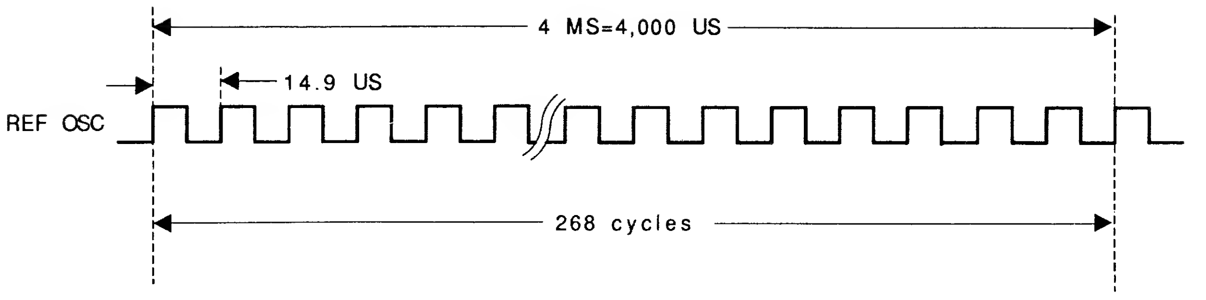


Figure 3-15 Refresh Time Periods

REF REQ is delayed approximately 145 ns to become DLY REF REQ, and then ANDed with ARRAY NOT BUSY to assert REF. ARRAY NOT BUSY is true when the arrays are free and can execute a refresh. The 145 ns delay insures that the MCL has had time to respond to BMAR SEND NO CMD and ARRAY NOT BUSY will not be negated by a new command after REF has been asserted.

REF asserts SET REF MODE, and then REF MODE via an OR gate. The assertion of REF MODE asserts REFRESH (MCL COLD START negated*) which is applied to the 4MBARRAY banks and places the banks into the refresh mode.

Approximately 100 ns after the assertion of SET REF MODE, CLK REFRESH asserts to the 4MBARRAY banks. CLK REFRESH initiates the RAS strobing sequence in the array banks. (The CAS strobe is not required because the arrays refresh a row at a time.) The 100 ns delay allows time for the 4MBARRAY banks to establish the refresh mode before the strobing sequence is started.

When the refresh operation has completed, array bank 1 asserts CLR REF REQ to the refresh logic. CLR REF REQ asserts PEND CLR REF REQ which is ANDed with REF to reset the REF REQ flip-flop and negate REF REQ. ANDing PEND CLR REF REQ with REF is required because CLR REF REQ is asserted by bank 1 every time it completes any operation (write, read, or refresh). The true state of REF selects only the CLR REF REQ associated with the refresh operation.

The negation of REF REQ negates SET SEND NO CMD and then BMAR SEND NO CMD allowing the MCL to send new commands to the MAR4 board.

* MCL COLD START refers to a power-up after a power outage that was too long for battery backup power to maintain the array data. (Battery backup power is maintains array data for ten minutes.) During a cold start, the 4MBARRAY banks go into a cold start mode which requires that REFRESH be negated (see Section 3.3.2.6).

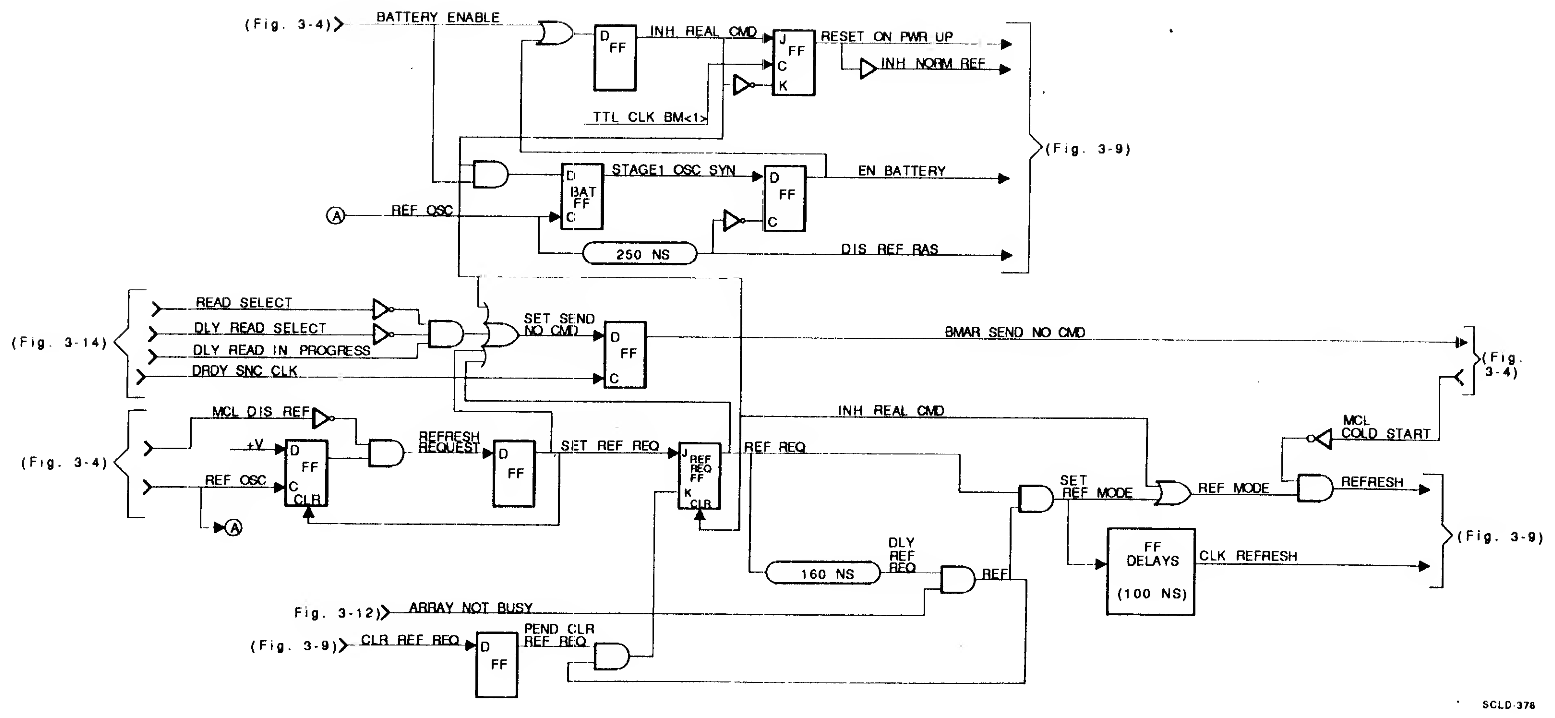


Figure 3-16 Refresh Block Diagram

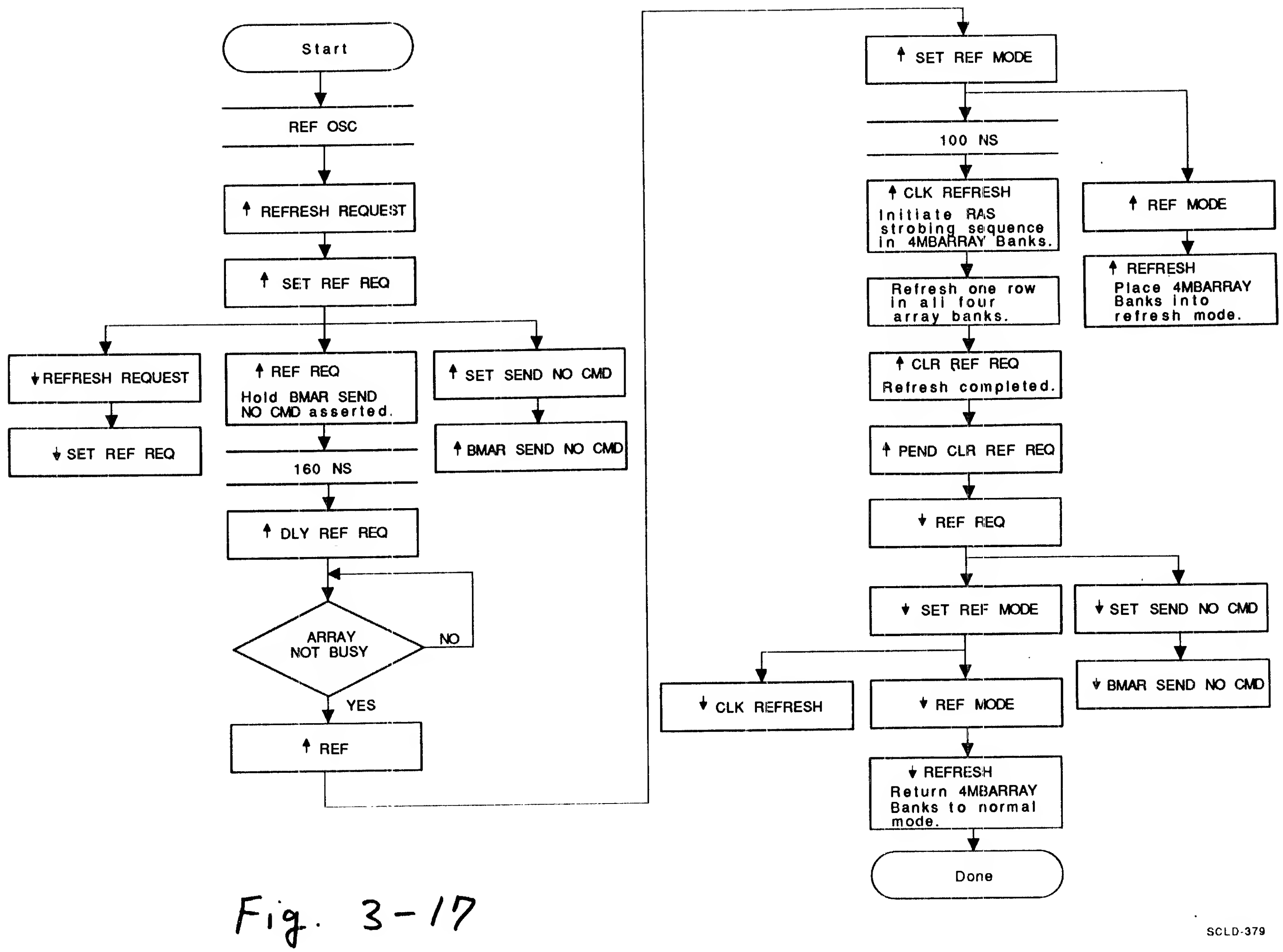


Fig. 3-17

Figure 3-17 Refresh Flow Diagram -- Normal Mode

In addition, the negated REF REQ bypasses the 145 ns delay and directly causes the negation of SET REF MODE, REF MODE, and REFRESH. This promptly takes the 4MBARRAY banks out of the refresh mode and returns them to normal operation.

3.3.6.2 Battery Mode -- When power is temporarily interrupted, the memory system automatically goes into battery mode. In battery mode, the arrays are continuously refreshed in order to retain the stored data until power is restored. Only refresh operations are allowed in battery mode. Read and write operations are suspended until normal power is resumed.

Battery mode refreshes do not use all of the refresh logic in the 4MBARRAY banks. In battery mode, only the barest minimum of logic is used and in its most efficient manner so as to draw the least amount of power from the battery.

Battery mode refreshes will execute reliably for a maximum of ten minutes. After that, the data stored in the arrays is no longer reliable.

The block diagram in Figure 3-16 illustrates the logic used in battery mode. Figure 3-18 is a flow diagram of the battery mode sequence.

Upon sensing an impending loss of power, the MCL:

- Sends no new commands to the MAR4.
- Inhibits new refresh cycles by asserting MCL DIS REF to the MAR4.
- Allows sufficient time (up to 675 ns) for any current operation to complete.
- Places the MAR4 into battery mode by asserting BATTERY ENABLE.

BATTERY ENABLE asserts INH REAL CMD which prepares the MAR4 for battery refreshes as follows:

- Asserts RESET ON PWR UP and INH NORM REF to the 4MBARRAY banks via a J-K flip-flop*. RESET ON PWR UP disables the normal refresh sequencing in the arrays. INH NORM REF disables the normal RAS refresh logic.

* The J-K flip-flop is set by system clock TTL CLK BM<1>. At the time battery mode is entered, power is still available to power the clock logic and place the MAR4 into battery mode. Once in battery mode, system clocks are not used

- Asserts REF MODE, via an OR gate, and then REFRESH. REFRESH places the arrays into the refresh mode.

- Asserts SET SEND NO CMD via an OR gate, and then BMAR SEND NO CMD. BMAR SEND NO CMD stays asserted during battery mode to assure an orderly transistion back to normal operation when power returns. That is, no new commands until battery mode is terminated.

- Holds the REF REQ J-K flip-flop reset thereby inhibiting any normal refreshes. This also is to ensure an orderly transition back to normal operation. That is, no new refreshes until battery mode is terminated.

- Enables a BAT flip-flop to be set by the next REF OSC cycle from the MCL. Battery power supplies the refresh oscillator logic in the MCL keeping the REF OSC signal enabled.

Figure 3-19 illustrates the timing involved in the initiation of battery mode refreshes. After INH REAL CMD asserts, the next rising edge of REF OSC clocks the BAT flip-flop asserting STAGE1

OSC SYN. REF OSC is delayed 250 ns to become DIS REF RAS. DIS REF RAS is inverted and clocks the next flip-flop causing EN BATTERY to assert. Due to the inverter, it is the trailing edge of DIS REF OSC that clocks the flip-flop and asserts EN BATTERY.

EN BATTERY and DIS REF RAS are applied to the 4MBARRAY banks along with REF OSC from the MCL. EN BATTERY enables the battery RAS logic while DIS REF RAS disables it. REF OSC is used in the 4MBARRAY banks to initiate the RAS strobing. As seen in the timing diagram, DIS REF RAS disables the battery RAS logic 250 ns after REF OSC has initiated the RAS strobing. This action results in the RAS logic being enabled for only 250 ns which is ample time for an array row to be refreshed. Disabling the RAS logic after 250 ns and keeping it disabled until the next REF OSC cycle, reduces the current drain in the arrays and conserves battery power.

As seen in Figure 3-19, EN BATTERY doesn't assert until after the trailing edge of REF OSC, hence the battery RAS logic is not enabled until the second REF OSC cycle after INH REAL CMD asserts. From the second cycle on, REF OSC initiates refresh cycles so long as EN BATTERY remains asserted.

When system power returns, the MCL negates MCL DIS REF and BATTERY ENABLE to return the MAR4 to normal operation. In going from battery to normal operation, one more battery refresh cycle is executed after BATTERY ENABLE negates and before control is returned to the normal refresh logic. This ensures that the battery logic has completed its last refresh before the normal refresh logic starts its first refresh.

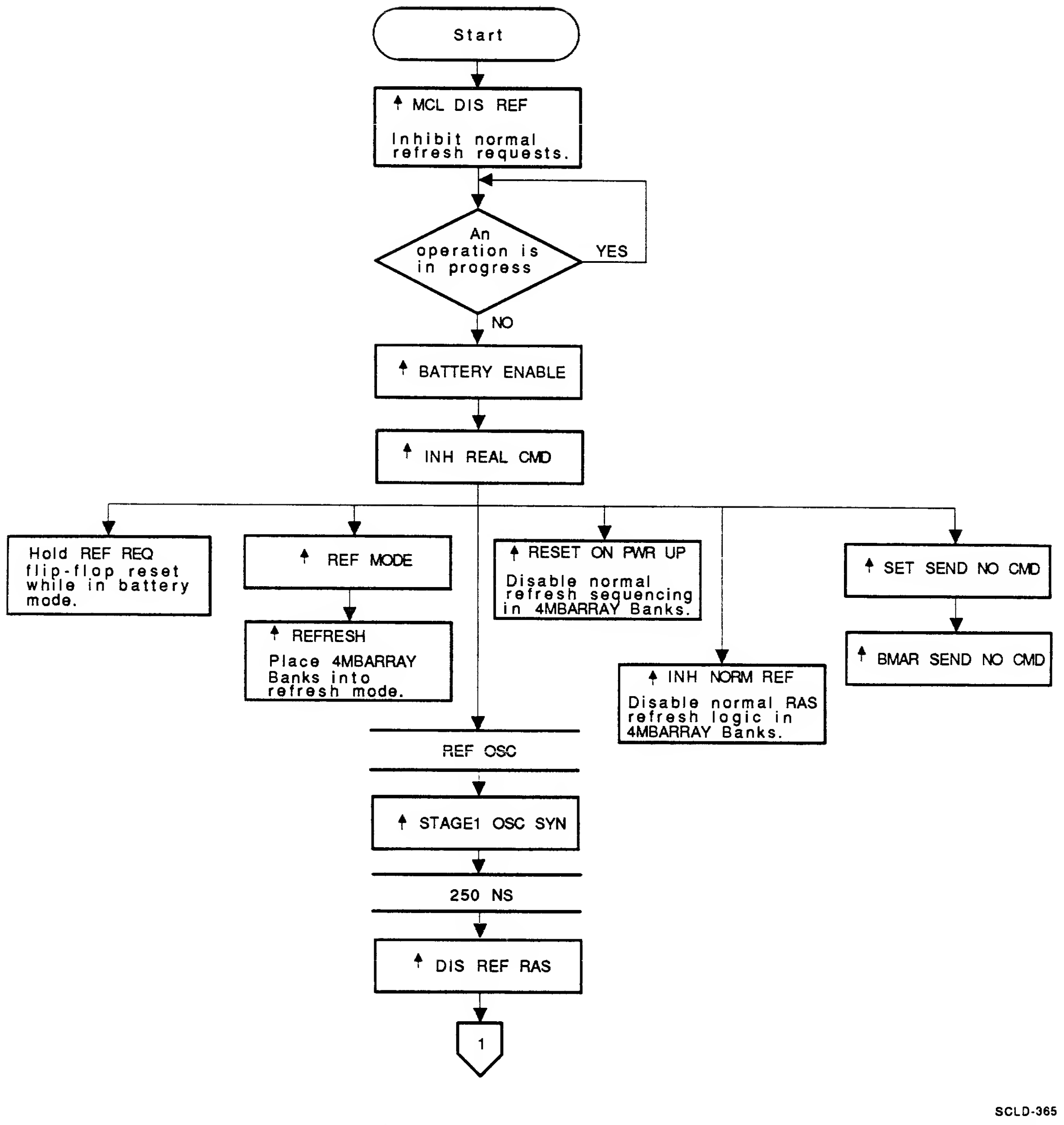

Start
↑ MCL DIS REF
Inhibit normal refresh requests.
An operation is in progress
YES
NO
↑ BATTERY ENABLE
↑ INH REAL CMD
Hold REF REQ flip-flop reset while in battery mode.
↑ REF MODE
↑ REFRESH
Place 4MBARRAY Banks into refresh mode.
↑ RESET ON PWR UP
Disable normal refresh sequencing in 4MBARRAY Banks.
↑ INH NORM REF
Disable normal RAS refresh logic in 4MBARRAY Banks.
↑ SET SEND NO CMD
↑ BMAR SEND NO CMD
REF OSC
↑ STAGE1 OSC SYN
250 NS
↑ DIS REF RAS
1
SCLD-365

Figure 3-18 Refresh Flow Diagram -- Battery Mode (Sheet 1 of 2)

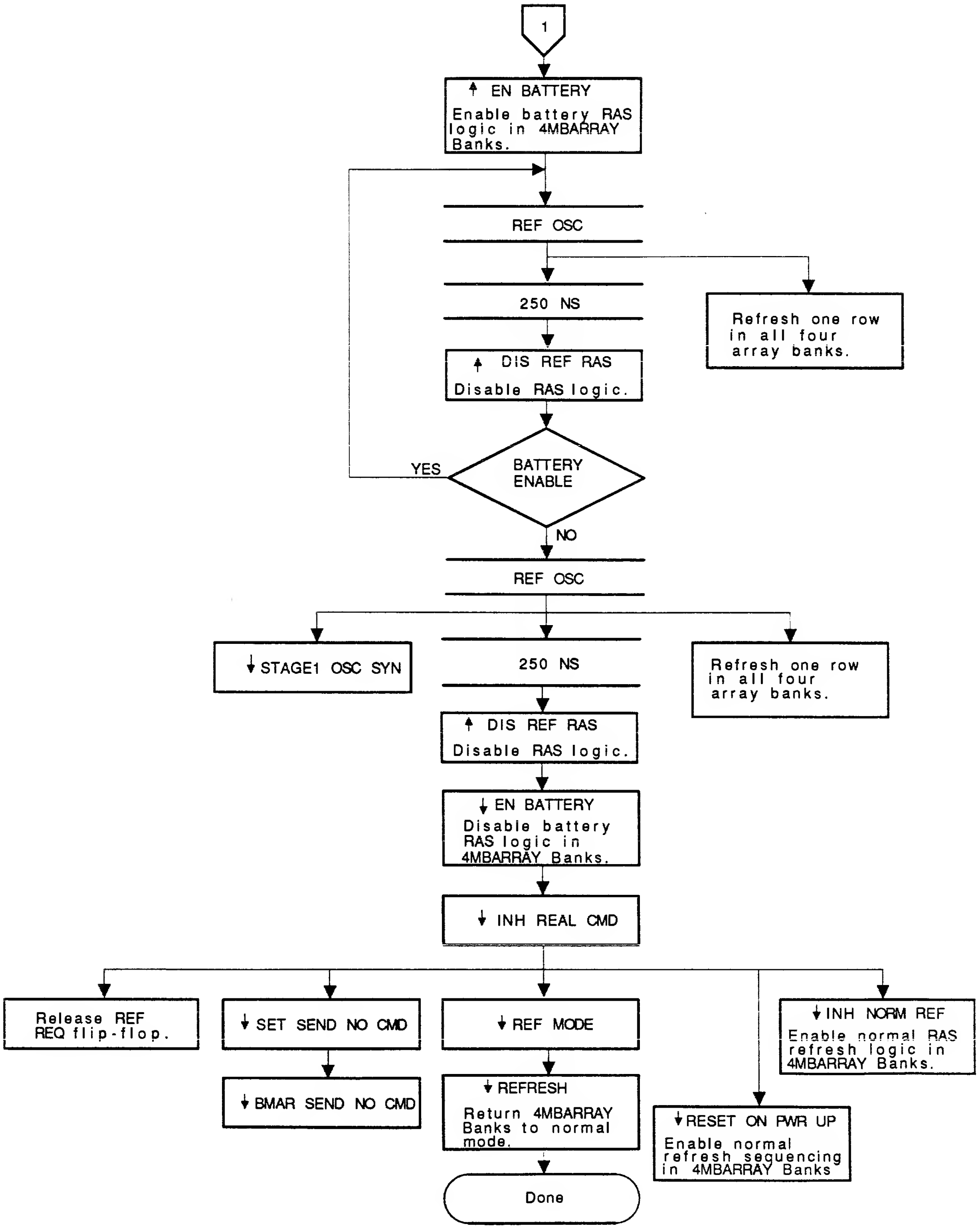

1
↑ EN BATTERY
Enable battery RAS logic in 4MBARRAY Banks.
REF OSC
250 NS
Refresh one row in all four array banks.
↑ DIS REF RAS
Disable RAS logic.
BATTERY ENABLE
YES
NO
REF OSC
↓ STAGE1 OSC SYN
250 NS
Refresh one row in all four array banks.
↑ DIS REF RAS
Disable RAS logic.
↓ EN BATTERY
Disable battery RAS logic in 4MBARRAY Banks.
↓ INH REAL CMD
Release REF REQ flip-flop.
↓ SET SEND NO CMD
↓ BMAR SEND NO CMD
↓ REF MODE
↓ REFRESH
Return 4MBARRAY Banks to normal mode.
Done
↓ RESET ON PWR UP
Enable normal refresh sequencing in 4MBARRAY Banks
↓ INH NORM REF
Enable normal RAS refresh logic in 4MBARRAY Banks.
SCLD-366

Figure 3-18 Refresh Flow Diagram -- Battery Mode (Sheet 2 of 2)

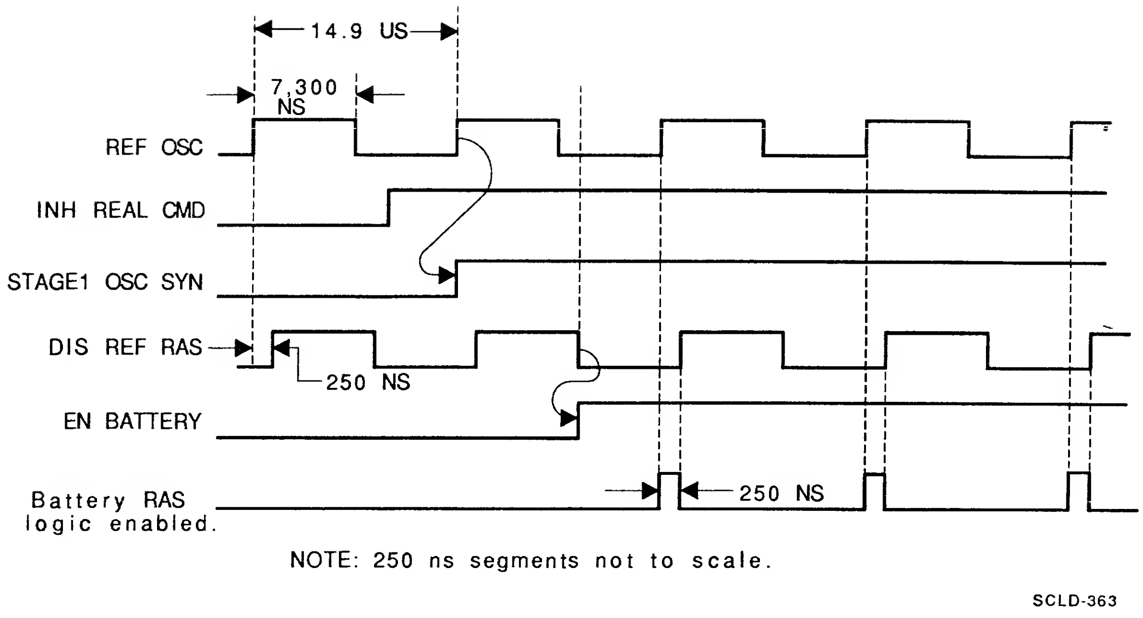


Figure 3-19 Initiation of Battery Mode Refreshes

Figure 3-20 illustrates the timing involved in the termination of battery mode refreshes. The negation of BATTERY ENABLE disables the BAT flip-flop. The next leading edge of REF OSC resets the BAT flip-flop (negating STAGE1 OSC SYN) while it initiates the last battery RAS strobing sequence in the arrays. The false state of STAGE1 OSC SYN results in EN BATTERY being negated by the next trailing edge of DIS REF RAS.

The negation of EN BATTERY disables the battery RAS logic in the arrays. It also causes the negation of INH REAL CMD which returns the MAR4 board back to normal operation by:

- Negating BMAR SEND NO CMD to the MCL to enable new commands to the MAR4.
- Releasing the REF REQ flip-flop allowing it to respond to normal refresh requests.
- Negating INH NORM REF to enable the normal RAS refresh logic in the 4MBARRAY banks.
- Negating RESET ON PWR UP to enable the normal refresh sequencing in the array banks.
- Negating REF MODE which in turn negates REFRESH which takes the 4MBARRAY banks out of the refresh mode.

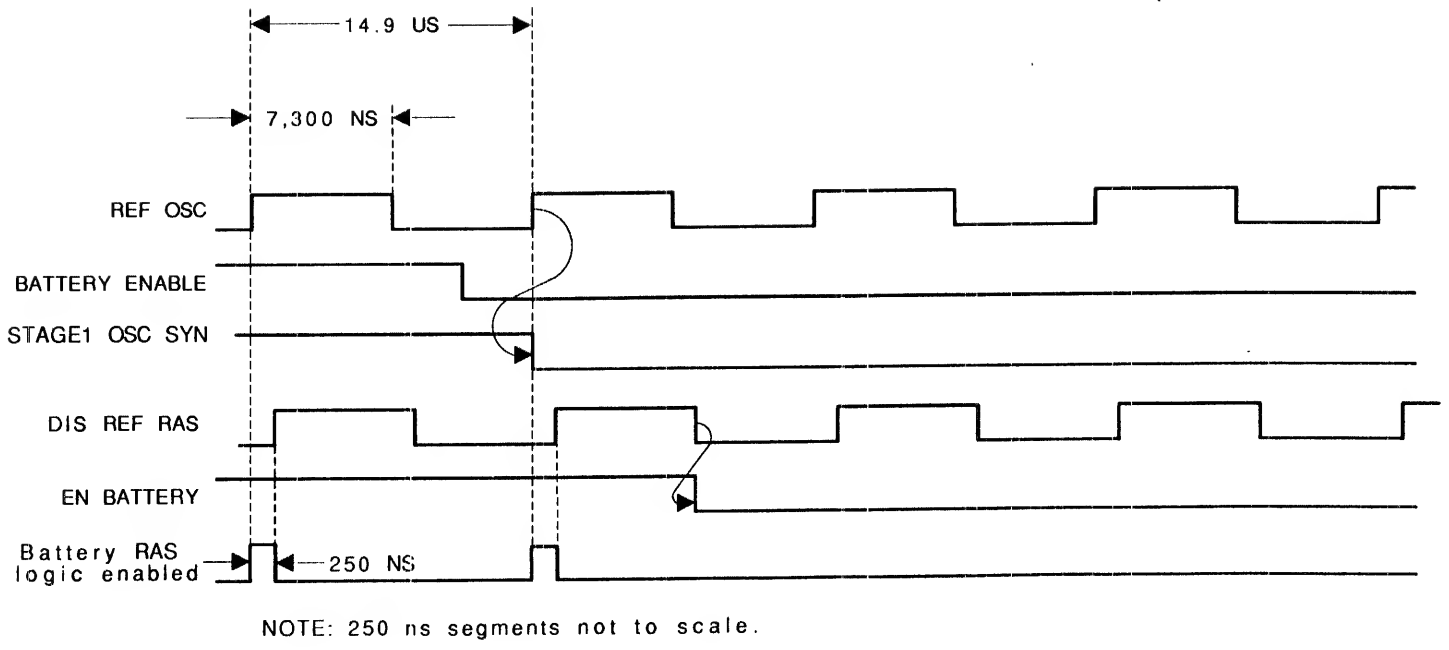


Figure 3-20 Termination of Battery Mode Refreshes

APPENDIX A
FLOW-DIAGRAM SYMBOLS

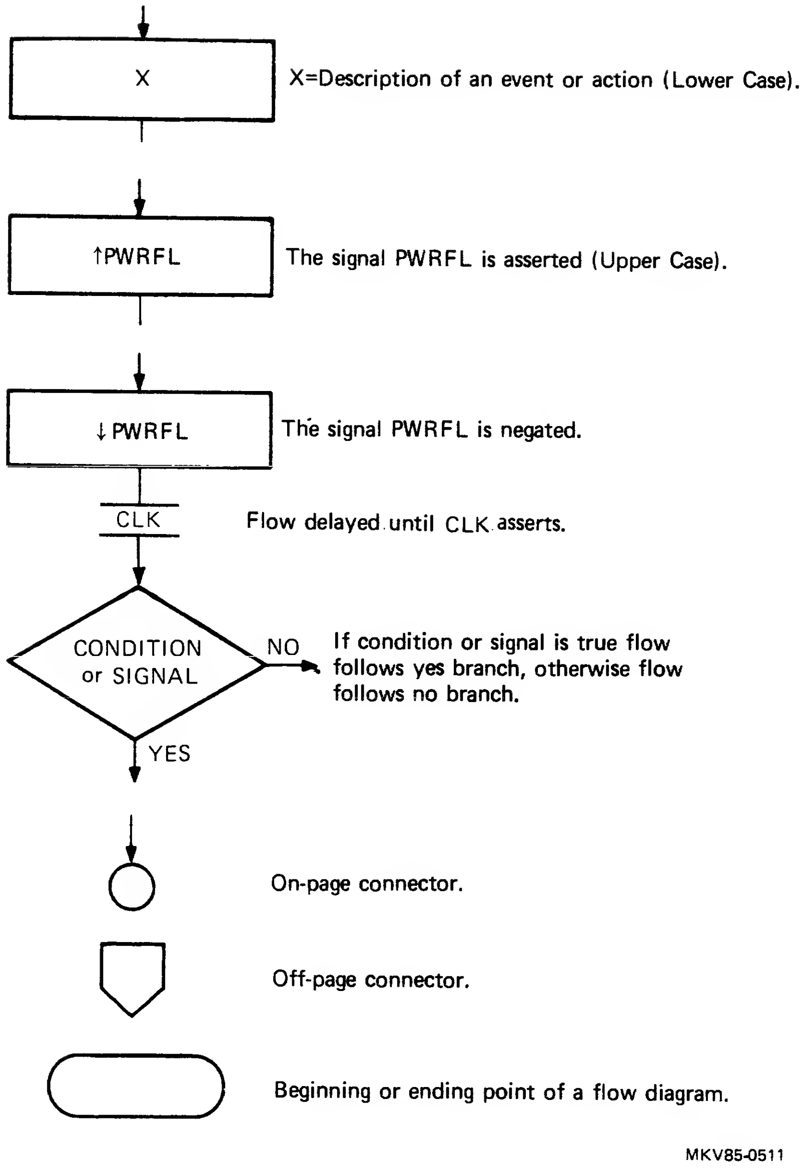

X
X=Description of an event or action (Lower Case).
↑PWRFL
The signal PWRFL is asserted (Upper Case).
↓PWRFL
The signal PWRFL is negated.
CLK
Flow delayed until CLK asserts.
CONDITION or SIGNAL
NO
YES
If condition or signal is true flow follows yes branch, otherwise flow follows no branch.
On-page connector.
Off-page connector.
Beginning or ending point of a flow diagram.
MKV85-0511

Figure A-1 Flow-Diagram Symbols

SECTION 10
NBI (NMI TO VAXBI ADAPTER)

CHAPTER 1
INTRODUCTION

1.1 GENERAL INFORMATION

The NMI-to-VAXBI (NBI) adapter interfaces the CPU and memory to the VAXBI-based I/O device controllers and adapters that are installed in the system. That is, it connects the CPU/memory backplane interconnect (the NMI) with up to two VAXBI buses. It does the following:

1. Reads/writes device registers - Allows the CPU, by means of NMI read/write transactions, to access the registers in the VAXBI devices. (The NBI initiates VAXBI read/write transactions in response to the NMI read/write transactions.)

2. Reads interrupt vectors - Allows the CPU, by means of NMI read transactions directed to vector registers in the NBI, to read interrupt vectors from the VAXBI devices. (The NBI initiates VAXBI IDENT transactions in response to the NMI read transactions.)

3. Reads/writes memory (DMA data transfers) - Allows the VAXBI devices, by means of VAXBI read/write transactions, to make DMA data transfers to/from the system's memory. (The NBI initiates NMI read/write transactions in response to the VAXBI transactions.)

4. Interrupts CPU - Allows the VAXBI devices to interrupt the CPU. (The NBI asserts the appropriate NMI interrupt lines in response to VAXBI INTR and IPINTR transactions.)

Device register and vector data transfers are one longword. DMA data can be transferred to/from a device in longword, quadword, or octaword blocks.

A single NBI consists of an NBIA and up to two NBIB modules as shown in Figure 1-1. The NBIA module plugs into a CPU backplane slot (connecting it to the NMI). Each NBIB module plugs into a VAXBI backplane slot (connecting it to a VAXBI). A VAXBI backplane, which also contains the VAXBI device controllers and adapters to be interfaced to the CPU, may be in the CPU cabinet or in an expansion cabinet. The connection between the NBIA and each NBIB (and thus between backplanes) is called the data bus. It consists of four ribbon cables that plug directly into the back (not the module side) of the CPU and VAXBI backplanes.

Up to two NBIs can be installed in some systems. These are systems having dual-CPU (21-slot) backplanes. Systems having a single-CPU (11-slot) backplane can accommodate only one NBI.

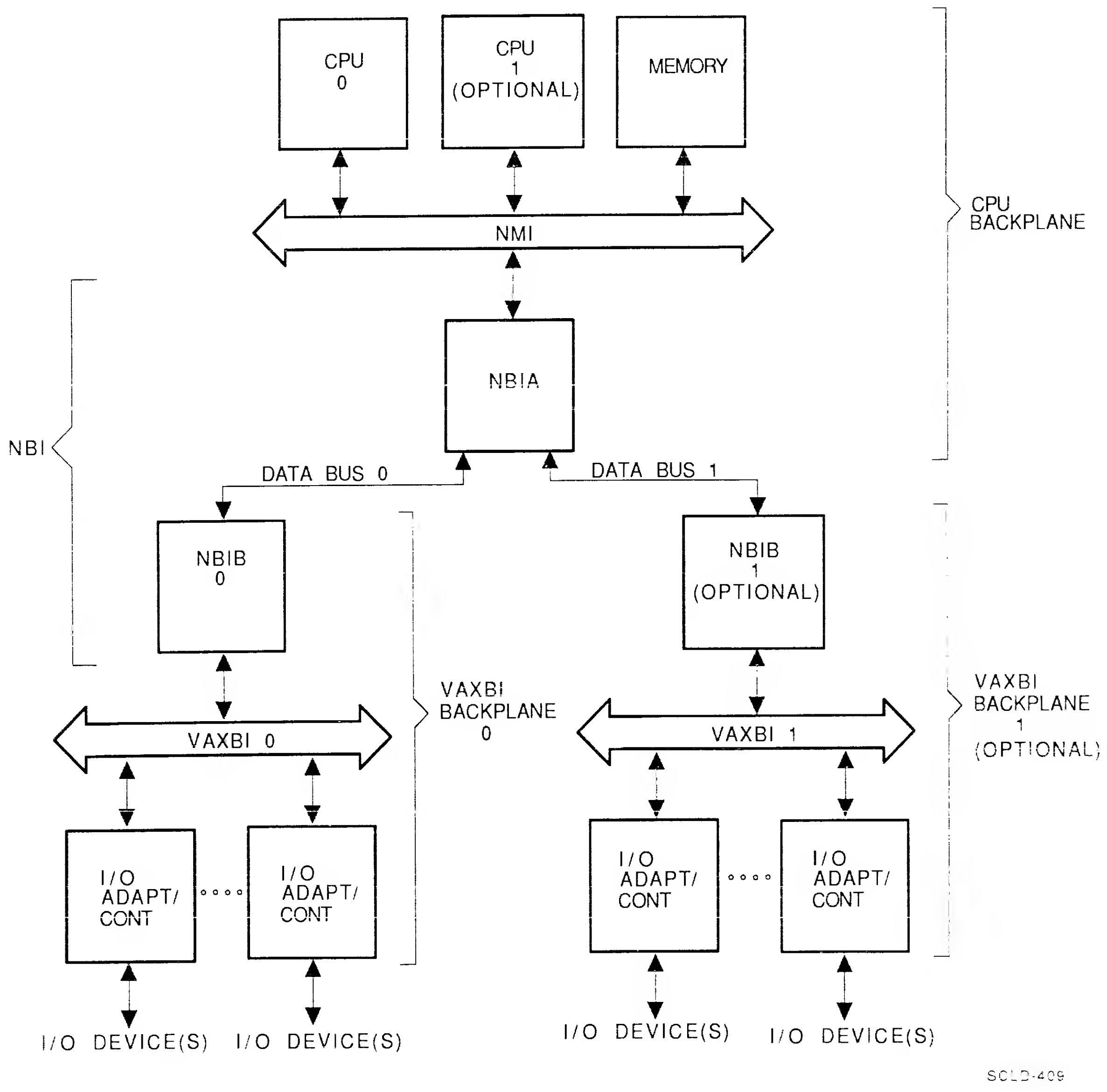


Figure 1-1 NBI Configuration

1.2 PHYSICAL DESCRIPTION AND CIRCUIT TECHNOLOGY

The NBIA (F1011) module is a 9-layer, 480-pin, extended-hex module that plugs into either slot 9 or 12 of dual-CPU backplanes, or in slot 2 of single-CPU backplanes. The NBIA module uses both ECL and TTL circuit components. The principal ECL components are 10KH MCAs. The principal TTL components are F544 transceiver latch circuits that interface to the data buses, and several PALs and FPLAs that are used for control purposes. Also, DC022 RAM chips are used in the NBIA's data path (in the transaction buffer). These RAMs have both an ECL and a TTL port.

An NBIB (T1020) module is a 10-layer, 300-pin, eurocard module that always plugs into slot 1 of the associated VAXBI backplane. The module components are mainly TTL. (Again, the principal components are F544 transceiver latches, PALs, and FLPAs.) To interface to the VAXBI, a custom ZMOS integrated circuit called the Bus Interconnect Interface Chip (BIIC) is used. This chip, along with a VAXBI clock driver chip and a VAXBI clock receiver chip, are located on a reserved area of the NBIB module called the "VAXBI corner".

1.3 BASIC BLOCK DIAGRAM

A block diagram of the NBI is shown in Figure 1-2. Basically, the NBIA consists of an ECL interface to the NMI, a TTL interface to the two data bus ports, and (to allow communication between these two interfaces), ECL/TTL translation logic and the DC022 transaction buffer that has both ECL and TTL ports. An NBIB consists basically of a TTL interface to its single data bus port, and a TTL interface to the VAXBI. The major component in the VAXBI interface is the BIIC.

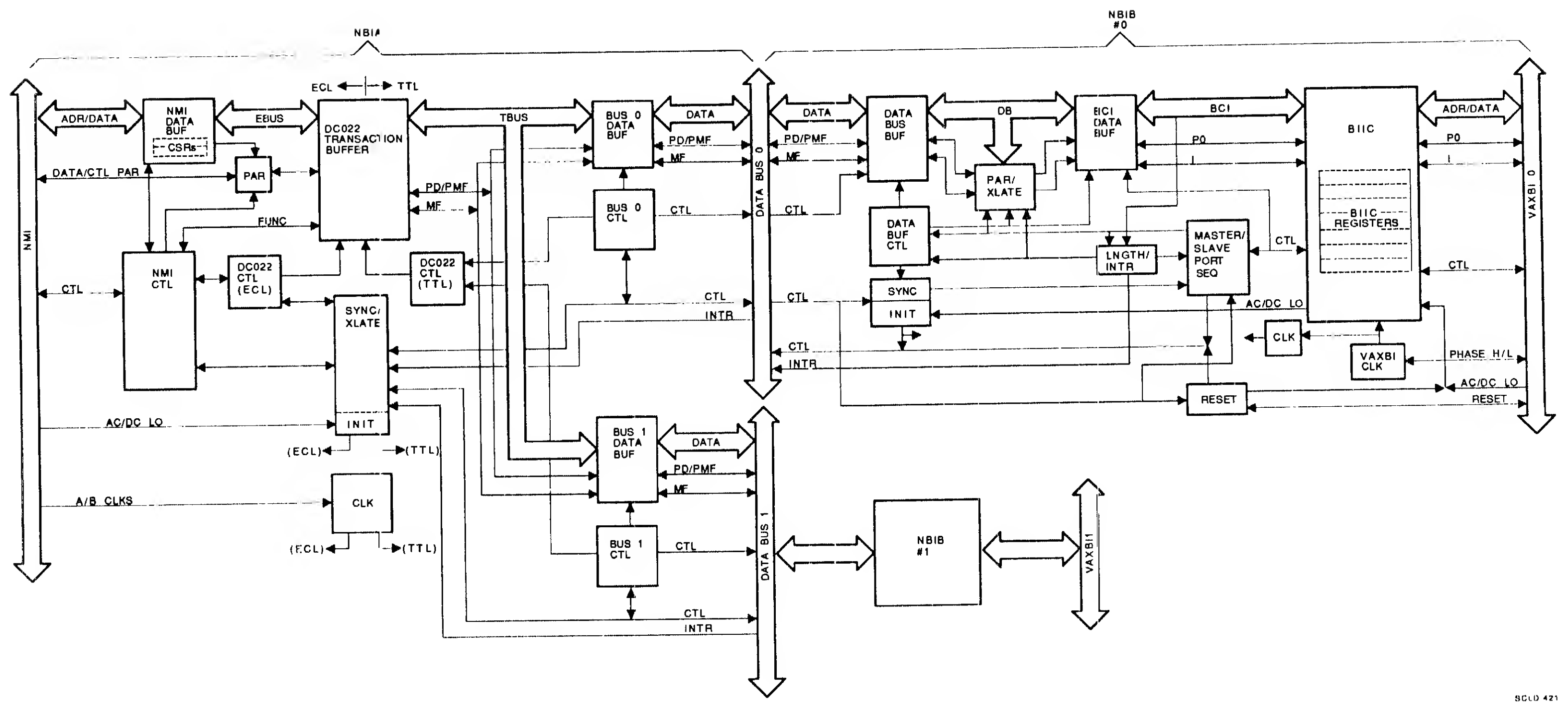


Figure 1-2 NBI Basic Block Diagram

The NBIA's data path consists of the following:

1. NMI data buffer -- Provides buffering for address/data transmitted and received during NMI transactions. Also contains the NBIA's control/status registers.

2. DC022 transaction buffer (Figure 1-3) -- A 16-location dual-port RAM that provides buffering for the command/address and data passed between the NMI and the data bus interfaces. The first five locations buffer the command/address and up to four longwords of read/write data for DMA transfers by VAXBI0 devices. A second group of five locations provide buffering for DMA transfers by VAXBI1 devices. (VAXBI0 is the first VAXBI connected to an NBI; VAXBI1 is the second.) Another two locations hold the command/address and the single longword of read/write data transferred when the CPU accesses a VAXBI device register. (The device may be on either VAXBI0 or VAXBI1.) Four transaction buffer locations are not used.

3. Data buffers -- Each buffer (one per data bus port) contains four sets of transceiver latches for buffering the command/address and data transmitted and received on the data bus.

The NBIB's data path consists of the following:

1. Data bus buffer -- Similar to the data buffers in the NBIA but contains eight sets of transceiver latches for buffering the command/address and data transmitted and received on the data bus.

2. BCI data buffer -- Contains two sets of latches for buffering the command/address and data transferred to/from the BIIC. (The interface to the BIIC is called the BCI.) The BCI data buffer is located between the data bus buffer and the BIIC.

3. BIIC -- Contains separate internal buffers for the command/address and the data transmitted on the VAXBI. Also, it contains VAXBI and BCI control/status registers, plus other registers for controlling (and showing the status of) the VAXBI operations performed by the BIIC.

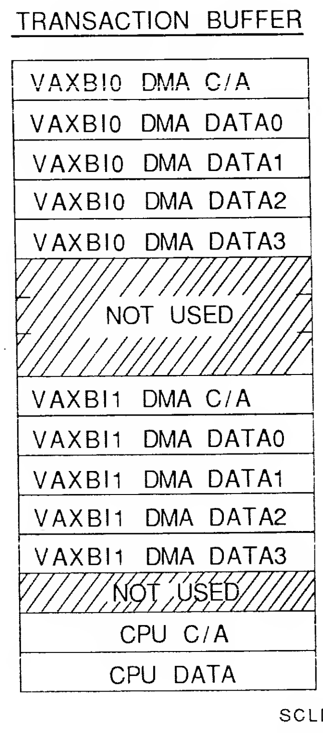


Figure 1-3 DC022 Transaction Buffer Organization

1.4 BASIC OPERATION

The basic operations performed by the NBI are CPU read/write data transfers, DMA read/write data transfers, and CPU interrupts.

1.4.1 CPU Read/Write Data Transfers

As stated previously, one of the basic functions of the NBI is to allow the CPU to read/write registers in the VAXBI devices. That is, the NMI read/write transaction initiated by the CPU (and received by the NBIA) causes a VAXBI read/write transaction by the NBIB. The registers accessed by the VAXBI transaction can be in any device connected to the VAXBI, and include the registers in the NBIB's own BIIC. The CPU can also access registers in the NBIA resulting in no VAXBI transactions. However, reading a vector register address in the NBIA can cause the NBIB to perform a VAXBI IDENT transaction. This, as stated previously, is to collect the vector from an interrupting VAXBI device.

Figure 1-4 shows the flow of command/address and data through the NBI during CPU read/write operations. Basic NMI and VAXBI transaction timing is also indicated.

When a VAXBI register is to be read or written, the command/address transmitted by the CPU (during the NMI transaction) is loaded into the NBIA's transaction buffer. If a VAXBI vector is to be read, a command/address consisting of an IDENT command and an interrupt level (both generated by control logic) is loaded into the transaction buffer. Also loaded in the transaction buffer, but only when a VAXBI register is to be written, is the write data transmitted by the CPU (during the NMI write transaction).

To complete a VAXBI register write operation in the NBI, both command/address and write data are read from the transaction buffer, sent to the NBIB over the data bus, and then transmitted on the VAXBI by the BIIC (during the VAXBI write transaction). If a VAXBI register or vector is to be read, only the command/address is transmitted on the VAXBI (during the VAXBI read transaction). When the device responds to the command/address, the register read data or vector transmitted on the VAXBI (by the device) is transferred from the BIIC to the NBIA where it is loaded in the transaction buffer. The data or vector is then read from the transaction buffer and transmitted on the NMI to end the read operation.

When a control/status register in the NBIA is read or written, the only transfer of data occurs within the NMI data buffer, which also contains the registers.

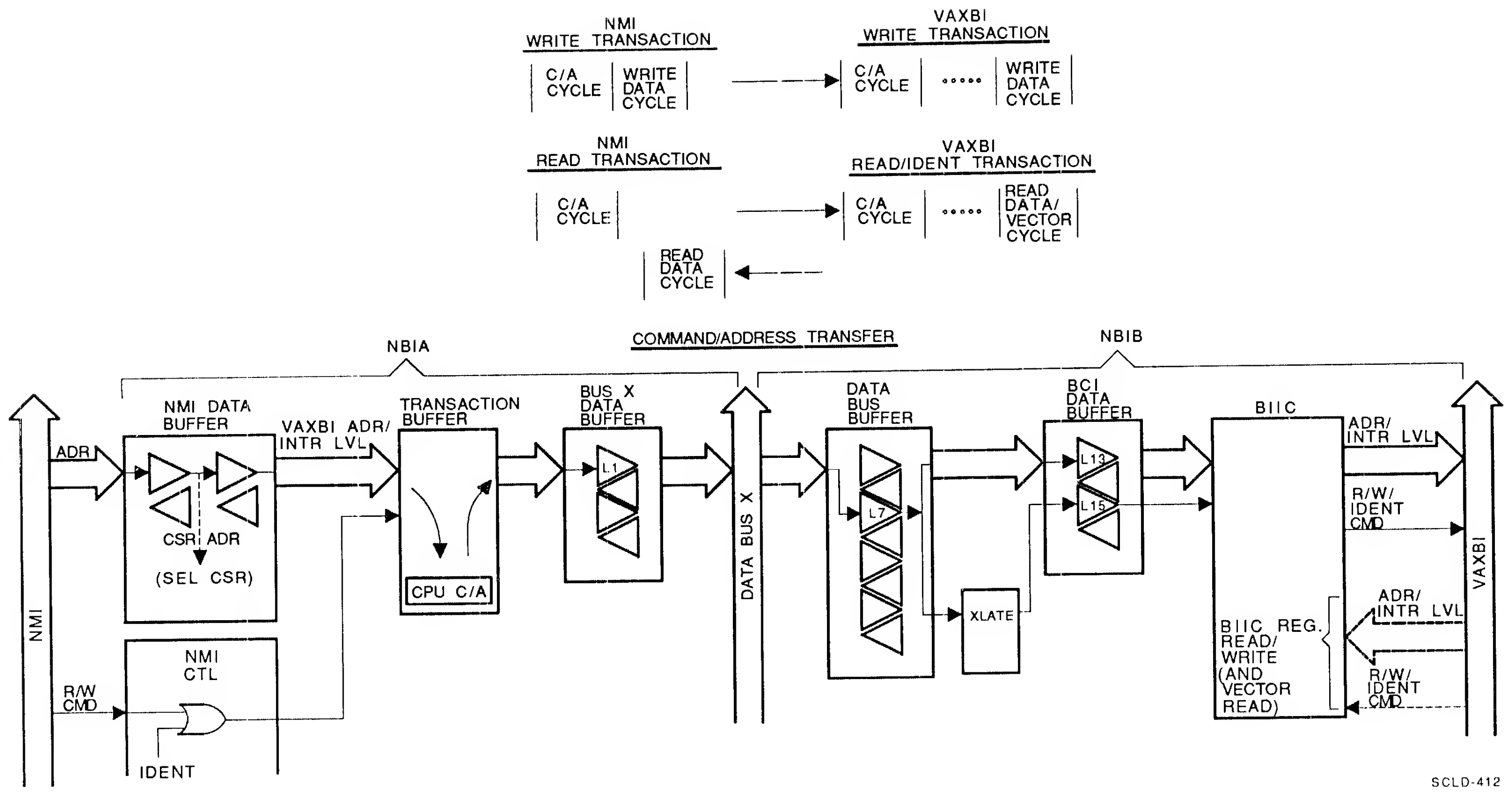


Figure 1-4 CPU Read/Write Data Transfer (Sheet 1 of 2)

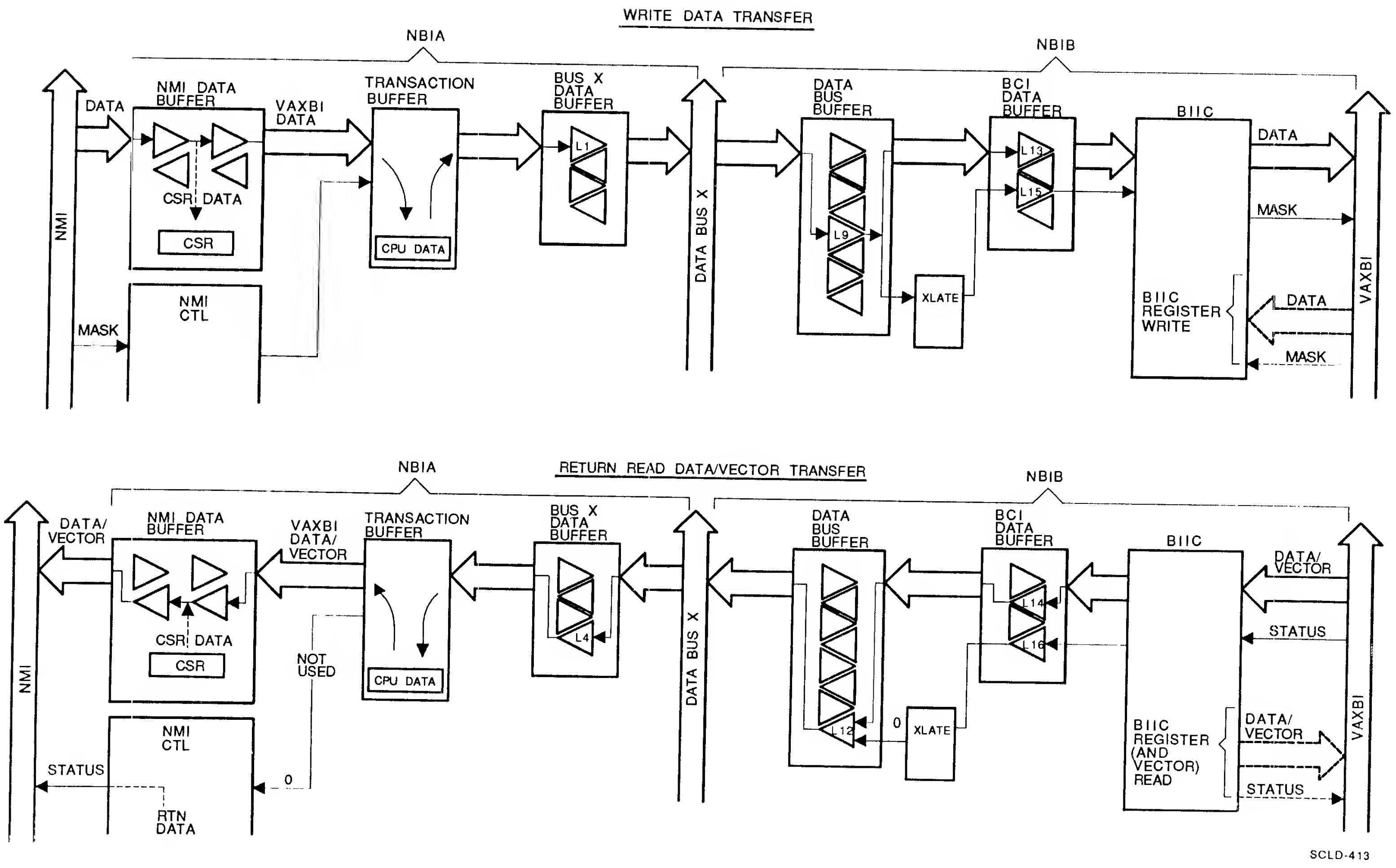


Figure 1-4 CPU Read/Write Data Transfer (Sheet 2 of 2)

1.4.2 DMA Read/Write Data Transfers

A VAXBI device can directly access the system's main memory (connected to the NMI) through the NBI. That is, a VAXBI read/write transaction initiated by the device and directed to a memory address causes the NBI to initiate the required NMI read/write transaction. (The memory address must fall within a range specified by starting and ending address registers in the BIIC.) The NMI transaction is the same length (longword, quadword, or octaword) as the VAXBI transaction with one exception. There is no NMI read quadword transaction, so a VAXBI read quadword transaction causes the NBI to do an NMI read octaword transaction. The addressed quadword in the octaword of data read from memory is then returned to the device.

Figure 1-5 shows the flow of command/address and data through the NBI during DMA read/write operations. During the VAXBI transaction, the command/address received by the BIIC is loaded in the NBIB's data bus buffer and then transferred to the NBIA's data buffer over the data bus. Also, if the transaction is a write, the write data (up to four longwords) is loaded in the NBIB's data buffer and transferred to the NBIA's data buffer. After its transfer to the NBIA, the command/address (and write data) is written in the transaction buffer and the NMI read or write transaction is initiated.

If the NMI transaction is a write, the command/address and write data are read from the transaction buffer and transmitted on the NMI to end the memory write data transfer in the NBI. If the NMI transaction is a read, and after the command/address is transmitted on the NMI, the memory returns the memory read data (up to four longwords) to the NBIA where it is written into the transaction buffer. Then the data is read from the transaction buffer, transferred to the NBIB over the data bus, and transmitted on the VAXBI by the BIIC to end the memory read data transfer in the NBI.

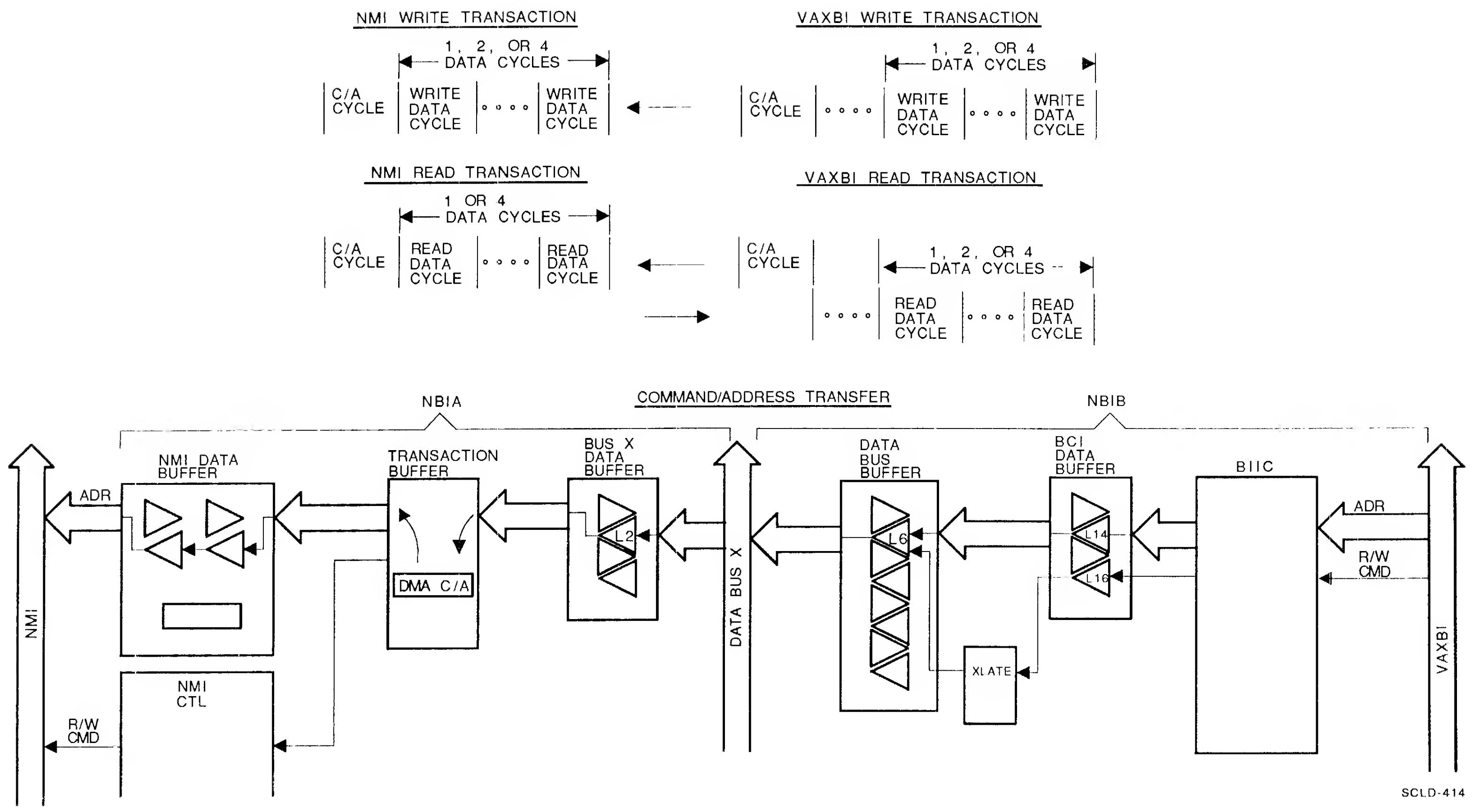


Figure 1-5 DMA Read/Write Data Transfers (Sheet 1 of 2)

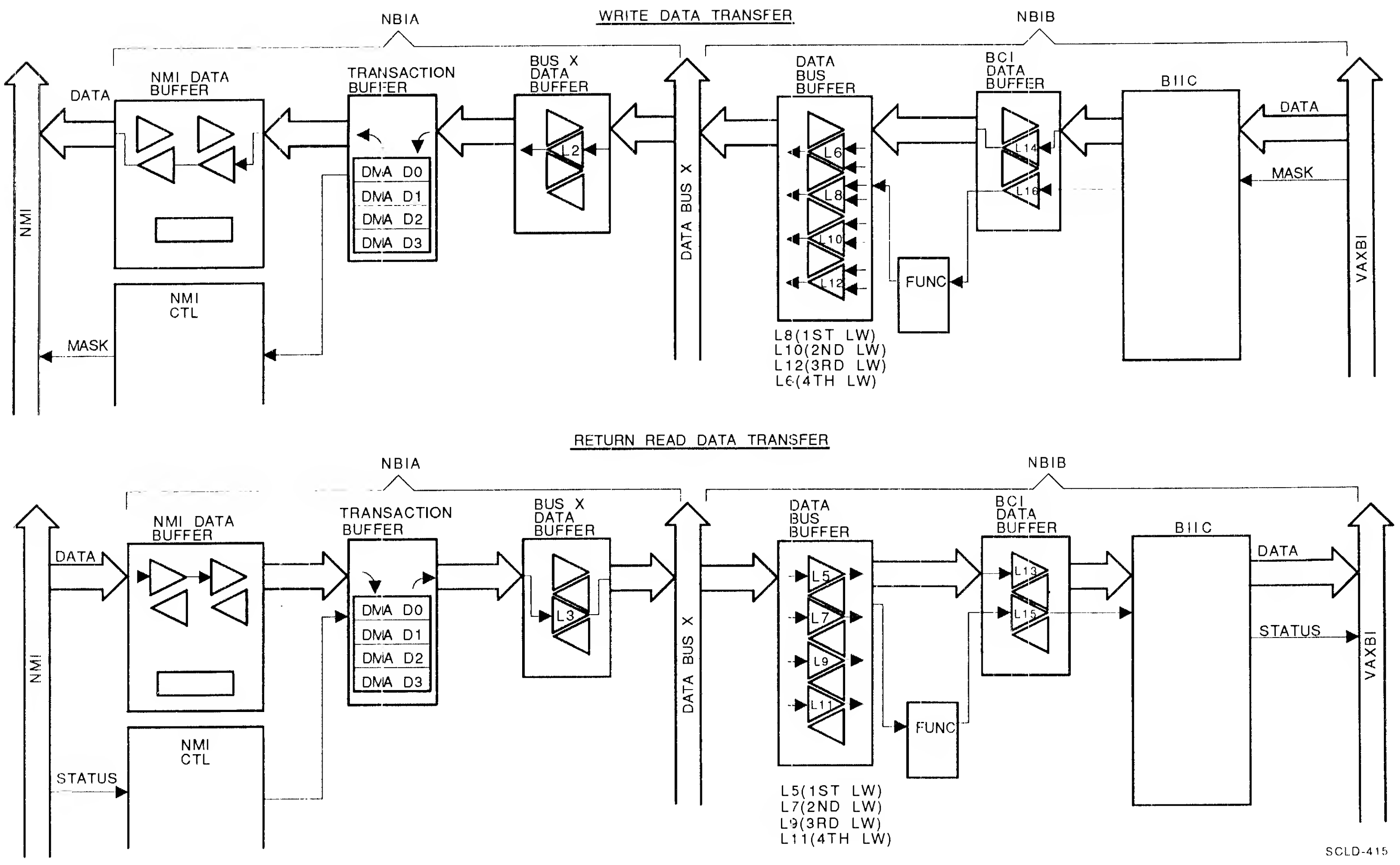


Figure 1-5 DMA Read/Write Data Transfers (Sheet 2 of 2)

1.4.3 Interrupt Operation

The NBI fields CPU interrupt requests by the VAXBI devices. The request may be due to a VAXBI INTR transaction (a conventional device interrupt request) or a VAXBI IPINTR transaction (an interprocessor interrupt request). An interprocessor interrupt request differs from a conventional interrupt request in that it is originated by one processor and directed to another. For example, an I/O processor on the VAXBI can direct an IPINTR transaction to the NBI and thus interrupt the CPU. Another difference is that no interrupt vector is read by means of a VAXBI IDENT transaction following the interrupt. That is, vector (and interrupt level) information is stored in the interrupted processor. Basic NBI operation in response to an INTR or IPINTR transaction is shown in Figure 1-6.

When the NBIB (its BIIC) receives a VAXBI INTR transaction, interrupt logic asserts one or more interrupt request signals on the data bus connecting to the NBIA. Normally, one interrupt request signal is asserted, but a single INTR transaction can specify up to four interrupt requests (BR<7:4>), each at a different priority level. (BR7 has the highest priority, BR4 the lowest.) When the NBIB receives a VAXBI IPINTR transaction, the interrupt logic forces a conventional interrupt request (an INTR transaction) by the NBIB's own BIIC. This request is at a BR4 level, and it asserts a request signal on the data bus like any other INTR transaction.

When the NBIA detects an asserted interrupt request on the data bus, it first resolves request priority (if more than one request is asserted) and then asserts an NMI interrupt request line (connected to the CPU) if it is not already asserting a higher priority request. Also, it asserts the encoded BR level on two other NMI signal lines to indicate request priority. The interrupt requests asserted on the NMI can be from either of the two NBIBs (and thus VAXBIs) that can be connected to the NBIA. Interrupt requests are also generated locally, by the NBIA itself. These (local) interrupt requests are at a BR4 level.

When the CPU is ready to service an NMI interrupt request, it reads a vector register address in the NBIA. There is a vector register corresponding to each interrupt priority (BR) level. As already described, when the interrupt is by a VAXBI device, a VAXBI IDENT transaction is executed by the NBIB and a device vector is returned to the CPU. (When the NBIB's BIIC is interrupting, it responds to its own IDENT and an internal vector is returned to the CPU.) If the interrupt is an NBI (local) interrupt, the BR4 vector register is read with no IDENT taking place and an NBI vector is returned to the CPU by the NBIA. All vectors read by the CPU are added to the system's SCBB to generate an address in the SCB. (The SCB location contains the dispatch address for the interrupt service routine.) There are three basic types of vectors, as shown in Figure 1-7.

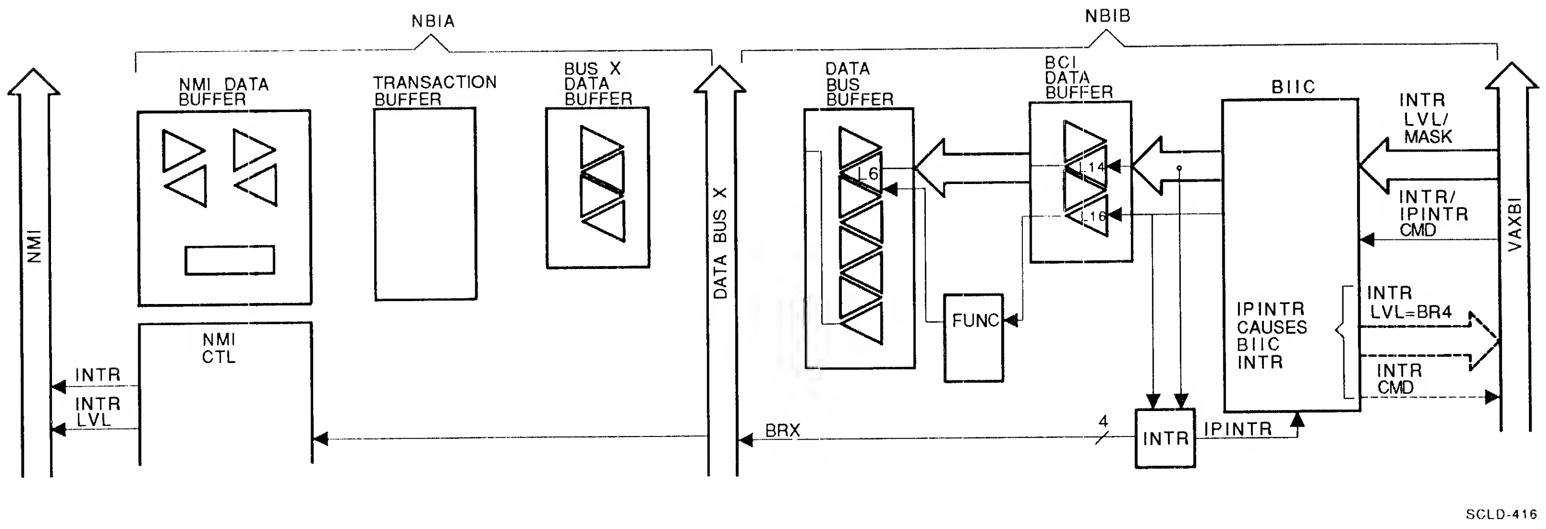


Figure 1-6 INTR/IPINTR Operation

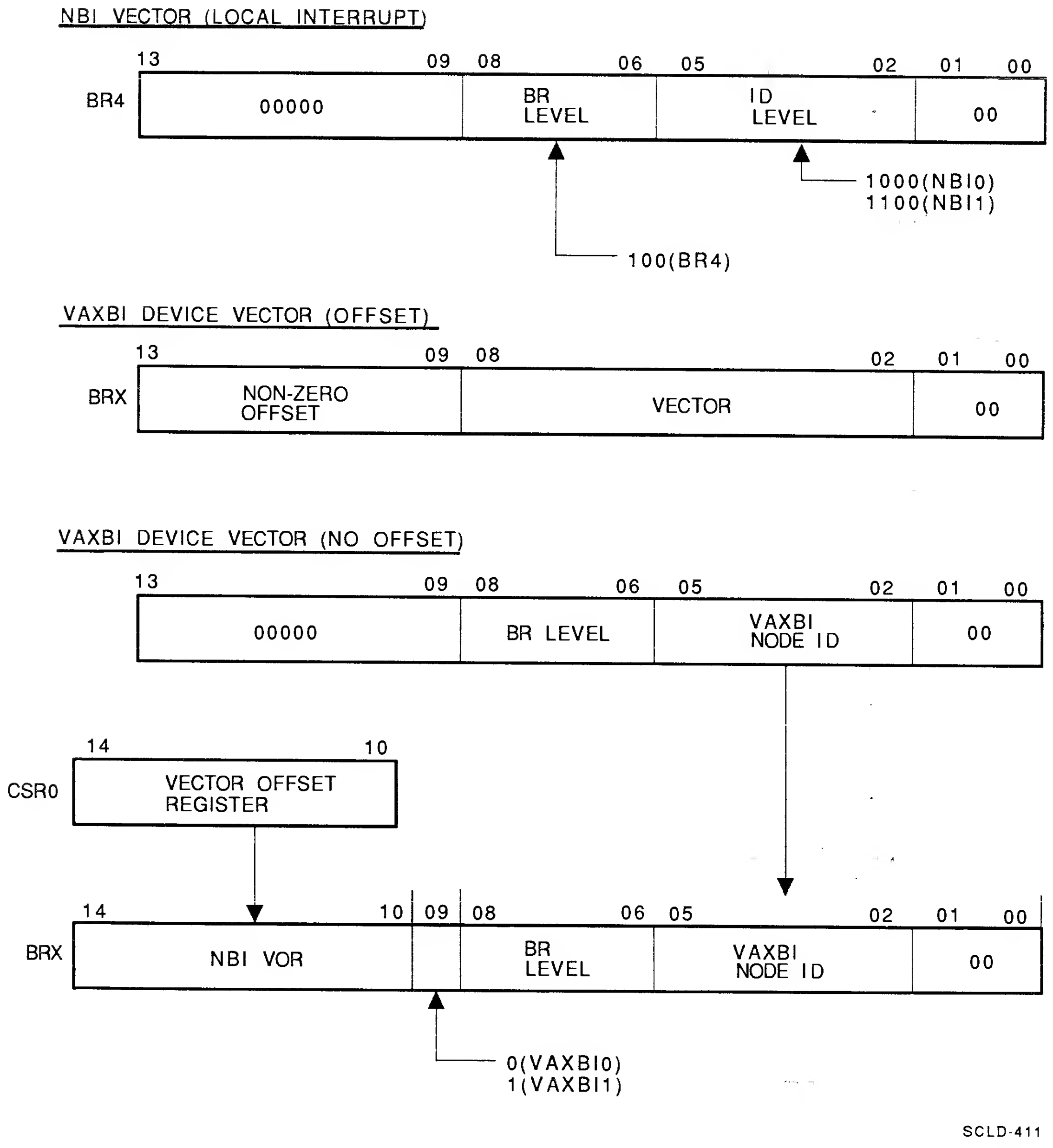


Figure 1-7 Interrupt Vector Format

The first type of vector is the NBI (local) interrupt vector. It has a value of 120 or 130 (hex) depending on which NBIA is asserting the vector. (As stated previously, two NBIs can be installed in some systems.) The NBI vector value is an SCB page 0 address. This is because page 0 is reserved for architectural and NMI device (nexus) vectors. (The NBIA is an NMI device.)

The other two types of vectors read by the CPU both consist of a device vector and a higher order (nonzero) vector offset. In one case, the offset (bits <13:9>) is supplied by the VAXBI device. In the other, the offset is supplied by the NBIA (when bits <13:9> from the device equal zero). The NBIA supplies the offset for most VAXBI devices. However, adapters such as the DWUBA (VAXBI to UNIBUS adapter) supply their own offset. The vectors sent with an adapter offset are from the devices interrupting on the other bus (the UNIBUS, in the case of the DWUBA).

Offset values are previously loaded in device and NBIA vector offset registers by the CPU. (The NBIA's vector offset register is in CSR0.) Offset values are chosen so that devices supplying an offset are allocated one page in the SCB per device, starting with page 1. Then, following these pages and starting with the next even page (this may leave an unused page), an SCB page is allocated for the devices (nodes) on each VAXBI not supplying an offset. Actually, this is all the devices, because an adapter like the DWUBA (which passes UNIBUS vectors to the CPU) can also interrupt as a VAXBI node (BIIC detected errors, etc.) and send an internal vector to the CPU.

An example of SCB page allocation for a system is shown in Figure 1-8. The system has two VAXBIs (pages 2 and 3) and one of the VAXBI nodes is a DWUBA (page 1).

	SCB PAGE
ARCHITECTURAL AND NMI NEXUS VECTORS	0
OFFSET (UNIBUS) DEVICE VECTORS FOR DWBUA	1
VAXBI0 NODE VECTORS	2
VAXBI1 NODE VECTORS	3

SCLD-417

Figure 1-8 SCB Format (Example)

1.5 NBI REGISTERS

Six registers in the NBIA and thirteen registers in the NBIB (in the BIIC) control and show the status of NBI operations. (The BIIC also contains four general-purpose registers not used in NBI operations.) In the register descriptions that follow, codes are used to indicate register bit read/write status and other information.

DCLOC	- Cleared when DCLO is asserted (NBIA registers) or when it goes from asserted to deasserted state (NBIB registers).
DCLOL	- Loaded when DCLO goes from asserted to deasserted state.
DCLOS	- Set when DCLO goes from asserted to deasserted state.
DMW	- BIIC diagnostic mode writeable.
INITC	- Cleared when ADAPTER INIT (in CSR1) is set.
RO	- Read-Only.
R/W	- Read/Write.
R/W1C	- Read/Write 1 to clear.
STOPC	- Cleared by VAXBI STOP command.
STOPS	- Set by VAXBI STOP command.
UNJAMC	- Cleared by UNJAM console command.

1.5.1 NBIA Registers

The registers in the NBIA (NMI nexus registers) are listed in Table 1-1. The registers are accessed by the CPU by means of NMI read/write (longword) transactions. Two register addresses are reserved and not currently used.

Table 1-1 NBIA Registers

Register	NMI Address (Hex)	Read/Write
Control/Status Register 0 (CSR0)	2X08 0000	R/W
Control/Status Register 0 (CSR1)	2X08 0004	R/W
Reserved (RSVD0)	2X08 0008	-
Reserved (RSVD1)	2X08 000C	-
BR4 Vector Register (BR4VR)	2X08 0010	RO
BR5 Vector Register (BR5VR)	2X08 0014	RO
BR6 Vector Register (BR6VR)	2X08 0018	RO
BR7 Vector Register (BR7VR)	2X08 001C	RO

Note: X = 0 (NBIA 0)
X = 4 (NBIA 1)

1.5.1.1 Control/Status Registers (CSR0 and CSR1) -- The NBIA has two control/status registers. Bit format is shown in Figures 1-9 and 1-10. Register bits are defined in Tables 1-2 and 1-3.

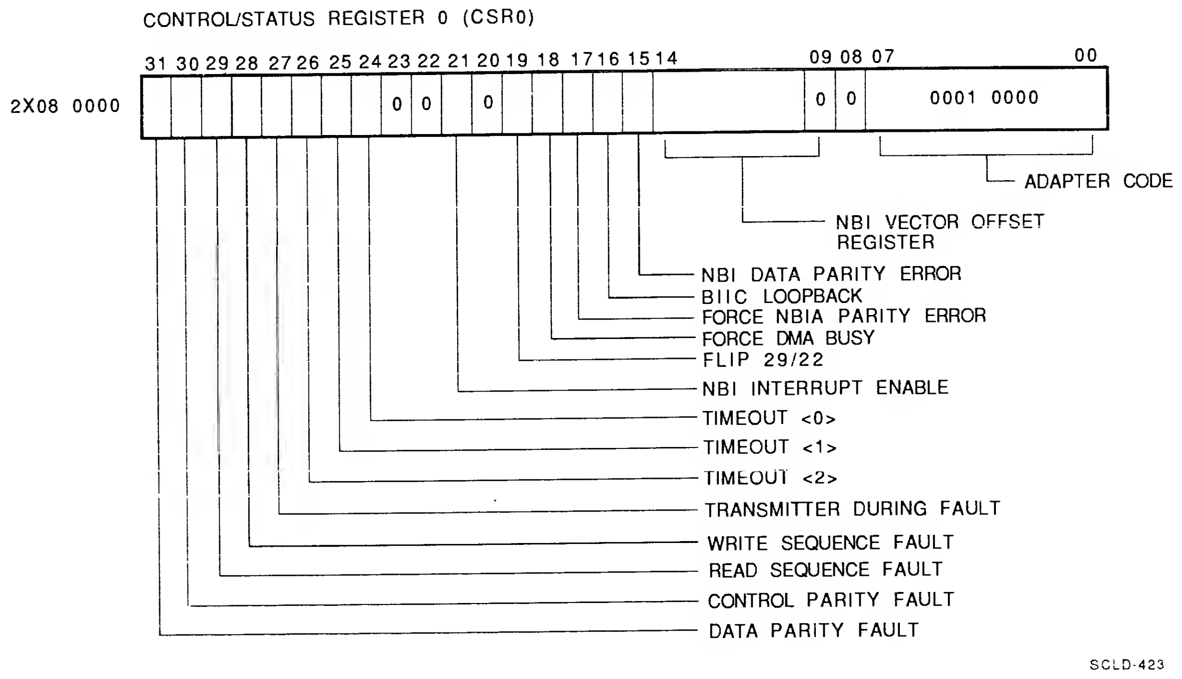


Figure 1-9 Control/Status Register 0 (CSR0)

Table 1-2 Control/Status Register 0 (CSR0) Bit Descriptions

Bit(s)	Name	Description
<31>	DATA PARITY FAULT (DPF)	Indicates bad parity detected for NMI ADDRESS DATA lines (RO, DCLOC, INITC).
<30>	CONTROL PARITY FAULT (CPF)	Indicates bad parity detected for NMI FUNCTION and MASK ID lines (RO, DCLOC, INITC).
<29>	READ DATA SEQUENCE FAULT (RDSF)	Indicates NBIA received incomplete or unexpected NMI return read data (RO, DCLOC, INITC).
<28>	WRITE DATA SEQUENCE FAULT (WDSF)	Indicates NBIA received incomplete NMI write data (RO, DCLOC, INITC).
<27>	TRANSMITTER DURING FAULT (TDF)	Indicates NBIA was transmitter when NMI fault was detected (RO, DCLOC, INITC).
<26:24>	TIMEOUT INTERRUPT	Indicate timeout condition detected by NBIA. Cause NBIA to generate CPU interrupt request (R/W1C, DCLOC, INITC). Writing any bit clears all three. TOI <2:0> / Timeout 000 None (No Interrupt) 001 Reserved 010 Reserved 011 Read Data Timeout 100 No Response Timeout 101 Bus Access Timeout 110 Interlock Busy Timeout 111 Busy Timeout
	Read Data Timeout	Command/address transmitted by NBIA to start NMI read transaction was acknowledged but no return read data was received.
	No Response Timeout	NBIA continued to receive no acknowledgment for the command/ address transmitted during retries of an NMI write transaction. No response was received during last retry.

Table 1-2 Control/Status Register 0 (CSR0) Bit Descriptions (Cont)

Bit(s)	Name	Description
	Bus Access Timeout	NBIA requested the NMI but was not granted use of the bus.
	Interlocked Busy Timeout	NBIA continued to receive no acknowledgment for the command/-address transmitted during retries of an NMI transaction. An INTERLOCKED response was received during last retry. (Cannot occur normally because an INTERLOCKED response does not normally cause a retry.)
	Busy Timeout	NBIA continued to receive no acknowledgment for the command/-address transmitted during retries of an NMI write transaction. A BUSY response was received during last retry.

NOTE
Timeouts occur after two system slow clock periods (approximately 6 ms at normal system clock rate).

Bit(s)	Name	Description
<23:22>	RESERVED	Not used (RO, ZEROS).
<21>	NBI INTERRUPT ENABLE (NIE)	Enables NBIA to make CPU interrupt requests (R/W, DCLOC, INITC, UNJAMC).
<20>	RESERVED	Not used (RO, ZERO).
<19>	FLIP 29/22 (FLIP)	Diagnostic data transfer control bit. Switches NMI address bits <29> and <22> in the NBIA causing a CPU read/write to a VAXBI address to generate a (DMA) read/write to a memory address (R/W, DCLOC, INITC, UNJAMC).
<18>	FORCE DMA BUSY (FDB)	Diagnostic control bit. Prevents DMA read/write operation from ending normally. Causes stall timeout error to be detected by NBIB's BIIC (R/W, DCLOC, INITC, UNJAMC).

Table 1-2 Control/Status Register 0 (CSR0)
Bit Descriptions (Cont)

Bit(s)	Name	Description
<17>	FORCE NBIA PARITY ERROR (FAPE)	Diagnostic control bit. Causes the NBIA to transmit bad data and control line parity on the NMI and both data buses (R/W, DCLOC, INITC, UNJAMC).
<16>	BIIC LOOPBACK (BILP)	Diagnostic control bit. Causes VAX read/write transactions generated by the NBIB's BIIC to be loopback requests (R/W, DCLOC, INITC, UNJAMC).
<15>	NBIA DATA PARITY ERROR (NDPE)	Indicates NBIA detected bad data parity for information read from its internal transaction buffer. Causes NBIA to generate CPU interrupt request (R/W1C, DCLOC, INITC).
<14:9>	NBI VECTOR OFFSET (NBIVO) <14:9>	Offset value for VAXBI device interrupt vectors not already offset. Bit <9> must be 0 (R/W, DCLOC, INITC).
<8>	RESERVED	Not used (RO, ZERO).
<7:0>	ADAPTER CODE (NAC) <7:0>	Indicate I/O adapter code. Equal to 10 (hex) for NBIA (RO).

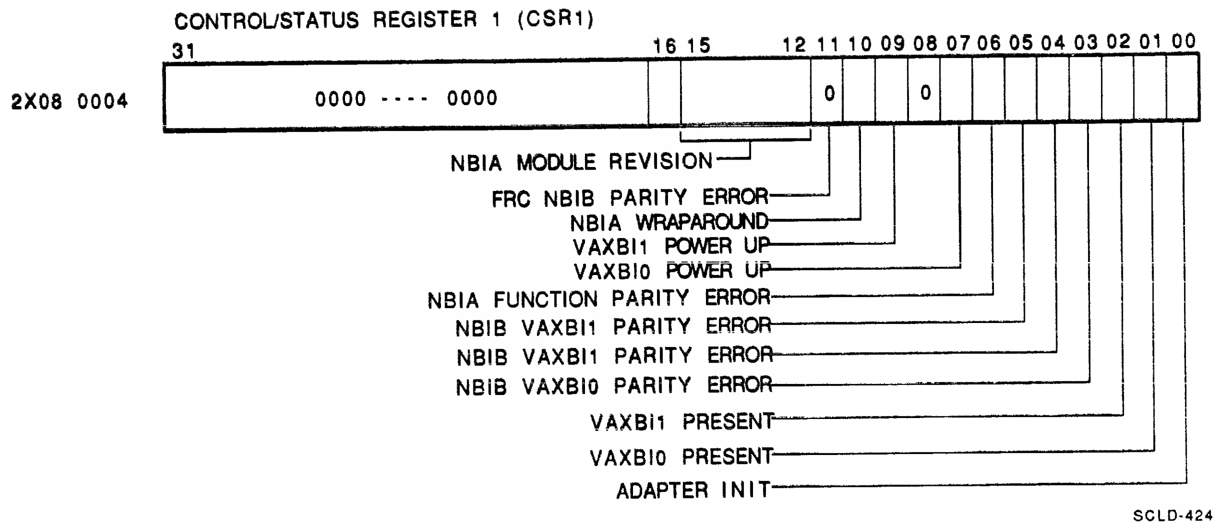


Figure 1-10 Control/Status Register 0 (CSR1)

Table 1-3 Control/Status Register 1 (CSR1) Bit Descriptions

Bit(s)	Name	Description
<31:16>	RESERVED	Not used (RO, ZEROS).
<15:12>	NBIA MODULE REVISION	NBIA module (hardware) revision level (RO).
<11>	RESERVED	Not used (RO, ZERO).
<10>	FORCE NBIB PARITY ERROR (FBPE)	Diagnostic control bit. Forces each NBIB to generate and detect bad parity in its internal data path (R/W, DCLOC, INITC, UNJAMC).
<9>	NBIA WRAPAROUND (NAWR)	Diagnostic control bit. Causes CPU read command/address to be looped back through the NBIA data bus buffers and returned to CPU as read data (R/W, DCLOC, INITC, UNJAMC).
<8>	RESERVED	Not used (RO, ZERO).

Table 1-3 Control/Status Register 1 (CSR1) Bit Descriptions (Cont)

Bit(s)	Name	Description
<7>	BI1 POWER UP (BI1PU)	Indicates DC LO and AC LO have been deasserted on VAXBI 1 after NBIB initialization. Causes NBIA to generate CPU interrupt request (R/W1C, DCLOC, INITC).
<6>	BI0 POWER UP (BI0PU)	Indicates DC LO and AC LO have been deasserted on VAXBI 0 after NBIB initialization. Causes NBIA to generate CPU interrupt request (R/W1C, DCLOC, INITC).
<5>	NBIA FUNCTION PARITY ERROR (NFPE)	Indicates NBIA detected bad control parity for information read from its internal trans- action buffer. Causes NBIA to generate CPU interrupt request (R/W1C, DCLOC, INITC).
<4>	NBIB BI1 PARITY ERROR (B0PE)	Indicates NBIB 1 detected bad parity in its internal data path. Causes NBIA to generate CPU interrupt request (R/W1C, DCLOC, INITC).
<3>	NBIB BI0 PARITY ERROR (B1PE)	Indicates NBIB 0 detected bad parity in its internal data path. Causes NBIA to generate CPU interrupt request (R/W1C, DCLOC, INITC).
<2>	BI1 PRESENT (BI1P)	Indicates NBIB 1 is connected to the NBIA and BI DC LO on the VAXBI is not asserted (RO, DCLOS).
<1>	BI0 PRESENT (BI0P)	Indicates NBIB 0 is connected to the NBIA and BI DC LO on the VAXBI is not asserted (RO, DCLOS).
<0>	ADAPTER INIT (ADIN)	Initializes NBIA, NBIB(s), and connected VAXBI devices. This bit is self-clearing (W0, DCLOC, INITC, UNJAMC).

1.5.1.2 **Vector Registers (BR4VR through BR7VR)** -- The NBIA has four vector registers: BR4VR, BR5VR, BR6VR, and BR7VR. (Refer to Figure 1-11.) When the NBIA is making a CPU interrupt request at some request level (BR4, BR5, BR6, or BR7), the CPU microcode reads the interrupt vector by reading the corresponding vector register. Vector format, which differs depending upon the type of interrupt request, is discussed in Section 1.4.3.

The vector registers are read-only registers. Zeros are returned (to the CPU) if an error is detected by the NBIB when reading a VAXBI vector. (The NBIB's BIIC posts an interrupt request to flag the error.) Also, zeros are returned if the interrupt request is no longer pending.

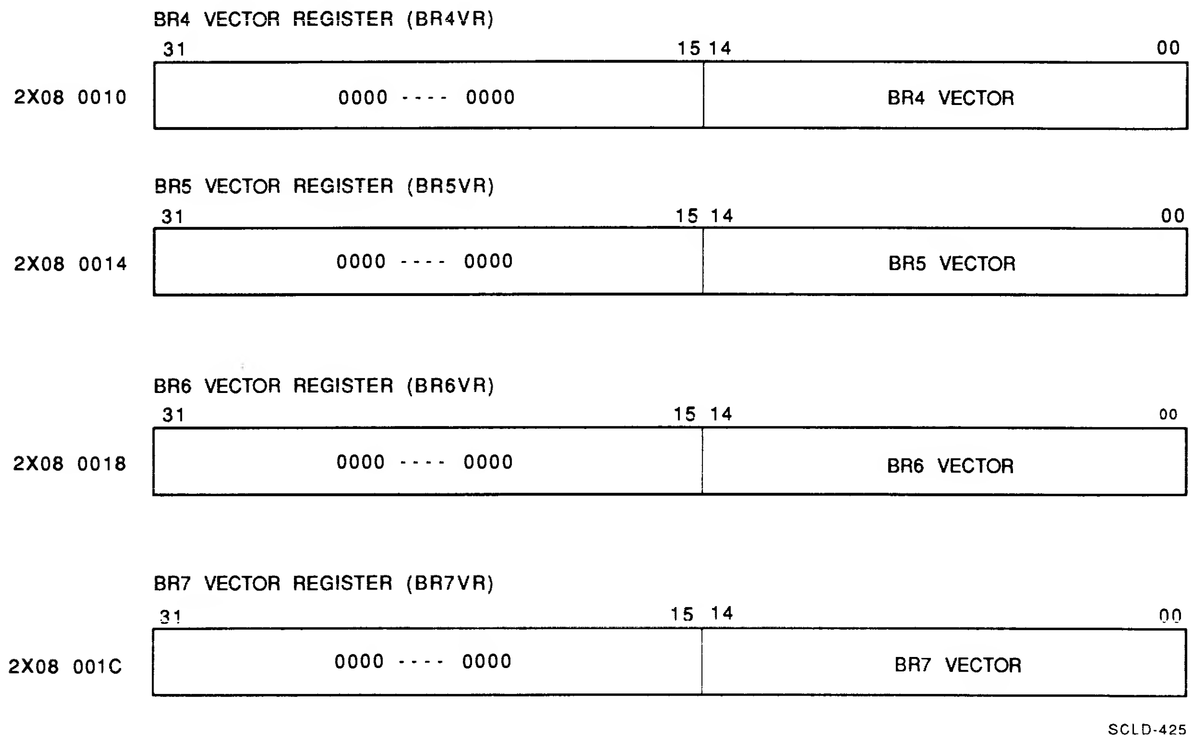


Figure 1-11 Vector Registers (BR4VR through BR7VR)

1.5.2 NBIB (BIIC) Registers

The NBIB registers are listed in Table 1-4. All are located in the NBIB's BIIC and are either VAXBI required registers that are implemented by all VAXBI nodes, or BIIC-specific registers that may or may not be implemented by a node. (The NBIB uses all the BIIC-specific registers.)

Like the NBIA registers, the NBIB's BIIC registers are accessed by the CPU by means of NMI read/write (longword) transactions. That is, the NMI read/write transactions cause VAXBI read/write transactions by the BIIC that access its own registers. The NBIB's BIIC registers can also be accessed by a connected VAXBI device (another VAXBI node) by means of VAXBI read/write transactions. This is not normally done, because the NBIB is the processor node on its associated VAXBI.

Table 1-4 NBIB (BIIC) Registers

Register	Address (Hex)	Read/Write
Device (DTYPE)	bb + 0	R/W
VAXBI Control/Status (BICSR)	bb + 4	R/W
Bus Error (BER)	bb + 8	R/W
Error Interrupt Control (EINTRCSR)	bb + C	R/W
Interrupt Destination (INTRDES)	bb + 10	R/W
IPINTR Mask (IPINTRMSK)	bb + 14	R/W
IPINTR/STOP Destination (FIPSDES)	bb + 18	R/W
IPINTR Source (IPINTRSRC)	bb + 1C	R/W
Starting Address (SADR)	bb + 20	R/W
Ending Address (EADR)	bb + 24	R/W
BCI Control/Status (BCICSR)	bb + 28	R/W
Write Status (WSTAT)	bb + 2C	R/W
Force IPINTR/STOP Command (FIPSCMD)	bb + 30	R/W
User Interrupt Control (UINTRCSR)	bb + 40	R/W
General-Purpose 0 (GPR0) - Not used	bb + F0	R/W
General-Purpose 1 (GPR1) - Not used	bb + F0	R/W
General-Purpose 2 (GPR2) - Not used	bb + F0	R/W
General-Purpose 3 (GPR3) - Not used	bb + F0	R/W

NOTE: Base address (bb) for NMI address:

bb = 2000 0000 (hex) + 2000 (hex) x node ID (NBIA 0, NBIB 0)
bb = 2200 0000 (hex) + 2000 (hex) x node ID (NBIA 0, NBIB 1)
bb = 2400 0000 (hex) + 2000 (hex) x node ID (NBIA 1, NBIB 0)
bb = 2600 0000 (hex) + 2000 (hex) x node ID (NBIA 1, NBIB 1)

Base address (bb) for VAXBI address:

bb = 2000 0000 (hex) + 2000 (hex) x node ID

1.5.2.1 Device Register (DTYPE) -- The device register (Figure 1-12) contains a revision code and a device type code to identify the VAXBI node. In the NBIB, the module revision code is jumper-selectable and automatically loaded by hardware at powerup. At the same time, a device type code of all 1s (a default value) is automatically loaded. Then, during system initialization, software loads a value that identifies the node as an NBIB. This register can be written by software when a BIIC selftest is in progress. This is necessary because the BIIC will not pass its selftest (a selftest occurs at powerup) with the default value of all 1s still loaded in the register. Register bit definitions are given in Table 1-5.

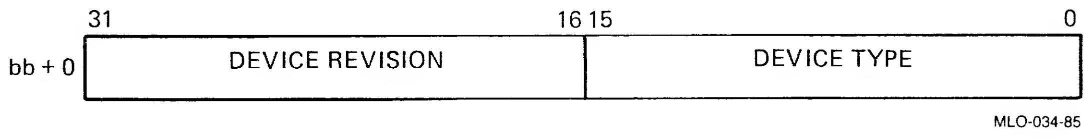


Figure 1-12 Device Register (DTYPE)

Table 1-5 Device Register (DTYPE) Bit Descriptions

Bit(s)	Name	Description
<31:16>	DEVICE REVISION	Indicates revision level of VAXBI device. NBIB module's revision code is loaded at powerup (R/W, DMW, DCLOL).
<15:0>	DEVICE TYPE	Indicates type of VAXBI node. All 1's are loaded at powerup in NBIB. The NBIB device code, equal to 0106 (hex), is then loaded by software during system initialization (R/W, DMW, DCLOL).

1.5.2.2 VAXBI Control/Status Register (BICSR) -- The VAXBI control/status register is shown in Figure 1-13. Bits are described in Table 1-6.

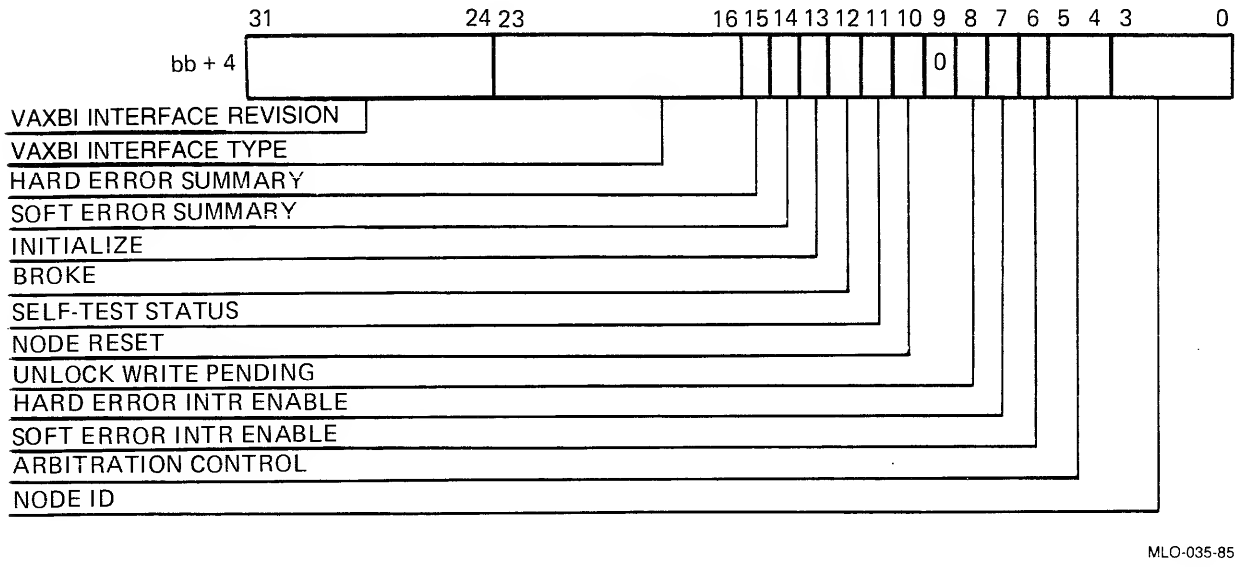


Figure 1-13 VAXBI Control/Status Register (BICSR)

Table 1-6 VAXBI Control/Status Register (BICSR) Bit Descriptions

Bit(s)	Name	Description
<31:24>	VAXBI INTERFACE REVISION (IREV)	Indicate the revision level of the primary interface to the VAXBI (the BIIC) (RO).
<23:16>	VAXBI INTERFACE TYPE (ITYPE)	Indicate the primary interface type. Equal to 01 (hex) for the BIIC (RO).
<15>	HARD ERROR SUMMARY (HES)	Indicates that one or more of the hard error bits in the bus error register is set (RO).
<14>	SOFT ERROR SUMMARY (SES)	Indicates that one or more of the soft error bits in the bus error register is set (RO).
<13>	INITIALIZE (INIT)	Implementation dependent. Not used in NBIB (R/W1C, DCLOS, STOPS).
<12>	BROKE	Indicates node has not yet passed its selftest. Only the BIIC performs a selftest in the NBIB (R/W1C, DCLOS).
<11>	SELF-TEST STATUS (STS)	Indicates the BIIC has passed its self-test (R/W, DCLOS).

Table 1-6 VAXBI Control/Status Register (BICSR) Bit Descriptions (Cont)

Bit(s)	Name	Description
<10>	NODE RESET (NRST)	Clears SELFTEST STATUS (bit <11>) and initiates BIIC selftest. Also causes BIIC's BCI DC LO output to be asserted, which initializes logic in NBIB. This bit is self-clearing (WO, DCLOC).
<9>	RESERVED	Not used (RO, ZERO).
<8>	UNLOCK WRITE PENDING (UWP)	Indicates the node (the NBIB) has completed a successful interlocked read (IRCI) transaction but not a subsequent lock write (UWMCI) transaction (R/W1C, DCLOC).
<7>	HARD ERROR INTR ENABLE (HEIE)	Enables the node (the NBIB) to generate an error interrupt when HARD ERROR SUMMARY (bit <15>) is set (R/W, DCLOC, STOPC).
<6>	SOFT ERROR INTR ENABLE (SEIE)	Enables the node (the NBIB) to generate an error interrupt when SOFT ERROR SUMMARY (bit <14>) is set (R/W, DCLOC, STOPC).
<5:4>	ARBITRATION CONTROL (ARB)	Determine node's (NBIB's) arbitration mode (R/W, DCLOC). ARB <5:4> — Arbitration Mode 00 — Dual round-robin 01 — Fixed high priority 10 — Fixed low priority 11 — Arbitration disabled When arbitration is disabled, node is inhibited from asserting D line corresponding to its node ID during arbitration cycles. Thus, it can never become bus master.
<3:0>	NODE ID	Indicate node's ID. NBIB's ID loaded at powerup and determined by ID plug inserted in backplane (RO, DMW, DCLOL).

1.5.2.3 Bus Error Register (BER) -- The bus error register (Figure 1-14) indicates any hard or soft errors detected by the BIIC during VAXBI transactions. Bits <30:16> are the hard error bits, bits <2:0> are the soft error bits. (Hard errors abort a transaction, soft errors do not.) The register also contains a bit that is not an error indicator. This is bit <3>, which indicates the BIIC's parity mode. Register bits are defined in Table 1-7.

Table 1-7 Bus Error Register Bit Descriptions

Bit(s)	Name	Description
<31>	RESERVED	Not used (RO, ZERO).
<30>	NO ACK TO MULTIRESPONDER COMMAND RECEIVED (NMR)	Indicates the bus master (the NBIB) received a NO ACK in response to a STOP, INTR, or IPINTR command (R/W1C, DCLOC).
<29>	MASTER TRANSMIT CHECK ERROR (MTCE)	Indicates the data transmitted by the VAXBI node (the NBIB) differed from that received. (The BIIC samples the data on the D, I, and P lines when it is the only source of transmitted data and compares it to the data it intended to transmit) (R/W1C, DCLOC).
<28>	CONTROL TRANSMIT ERROR (CTE)	Indicates the node (the NBIB) detected that NO ARB, BUSY, or CNF <2:0> was deasserted when the node was attempting to assert the signal (R/W1C, DCLOC).
<27>	MASTER PARITY ERROR (MPE)	Indicates the node (the NBIB), when bus master, detected a parity error on the VAXBI during a read (or vector) ACK data cycle (R/W1C, DCLOC).
<26>	INTERLOCK SEQUENCE ERROR (ISE)	Indicates a node (the NBIB) successfully completed a UWMCI transaction when no corresponding IRCI transaction has been issued previously (R/W1C, DCLOC).
<25>	TRANSMITTER DURIONG FAULT (TDF)	Indicates the node (the NBIB) detected a parity error during a cycle when it was responsible for transmitting correct parity (R/W1C, DCLOC).

Table 1-7 Bus Error Register Bit Descriptions (Cont)

Bit(s)	Name	Description
<24>	IDENT VECTOR ERROR (IVE)	Indicates a node (the NBIB) did not receive an ACK response to an interrupt vector it transmitted on the VAXBI during an IDENT (R/W1C, DCLOC).
<23>	COMMAND PARITY ERROR (CPE)	Indicates the node (the NBIB) detected a parity error during a command/address cycle (R/W1C, DCLOC).
<22>	SLAVE PARITY ERROR (SPE)	Indicates the node (the NBIB), when bus slave, detected a parity error during a write ACK or STALL data cycle or during a BDCST ACK data cycle (R/W1C, DCLOC).
<21>	READ DATA SUBSTITUTE (RDS)	Indicates the node (the NBIB), when bus master, received a READ DATA SUBSTITUTE or RESERVED code during a read (or vector) data cycle and no parity error was detected (R/W1C, DCLOC).
<20>	RETRY TIMEOUT (RTO)	Indicates the node (the NBIB), when bus master, received 4096 consecutive RETRY responses from the slave for the same transaction (R/W1C, DCLOC).
<19>	STALL TIMEOUT (STO)	Indicates the BIIC's slave port asserted the STALL code on the BCI RS <1:0> lines for 128 consecutive cycles (R/W1C, DCLOC).
<18>	BUS TIMEOUT (BTO)	Indicates the node (the NBIB) could not start a pending bus transaction before 4096 bus cycles elapsed (R/W1C, DCLOC).
<17>	NONEXISTENT ADDRESS (NEX)	Indicates the node (the NBIB), when bus master, detected a NO ACK response to a read/write-type transaction and no command parity or master transmit check error was detected (R/W1C, DCLOC).
<16>	ILLEGAL CONFIRMATION ERROR (ICE)	Indicates the node (the NBIB), when bus master or slave, received an illegal or RESERVED confirmation code (R/W1C, DCLOC).

Table 1-7 Bus Error Register Bit Descriptions (Cont)

Bit(s)	Name	Description
<15:4>	RESERVED	Not used (RO, ZEROS).
<3>	USER PARITY ENABLE (UPEN)	Indicates the user interface to the BIIC (the NBIB logic), not the BIIC, is to generate VAXBI parity (RO, DCLOL).
<2>	ID PARITY ERROR (IPE)	Indicates the node (the NBIB) detected a parity error on the VAXBI I lines during an embedded arbitration cycle (R/W1C, DCLOC).
<1>	CORRECTED READ DATA (CRD)	Indicates the node (the NBIB), when bus master, received a CORRECTED READ DATA status code during a data cycle and no parity error was detected (R/W1C, DCLOC).
<0>	NULL BUS PARITY ERROR (NPE)	Indicates odd parity was detected on the VAXBI during the second cycle of a two-cycle sequence during which VAXBI NO ARB and VAXBI BSY were not asserted. (R/W1C, DCLOC)

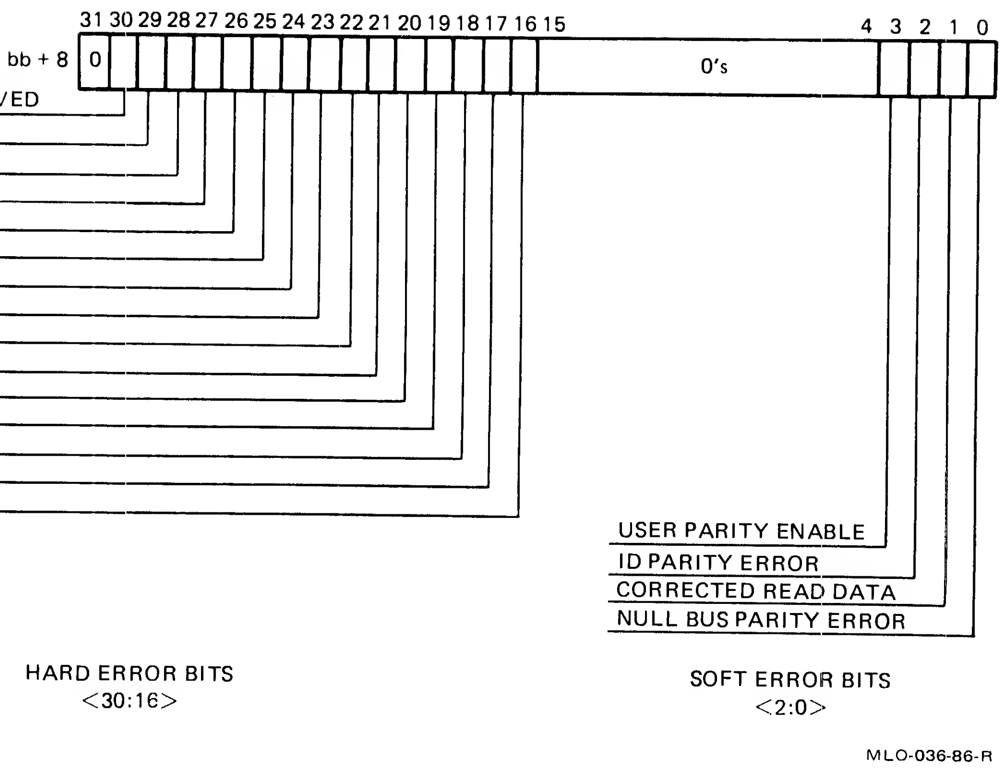


Figure 1-14 Bus Error Register (BER)

1.5.2.4 Error Interrupt Control Register (EINTRCSR) -- The error interrupt control register controls the operation of interrupts initiated by a BIIC-detected bus error or when an interrupt is forced by setting bit <20>. Register bit format is shown in Figure 1-15. Bit definitions are given in Table 1-8.

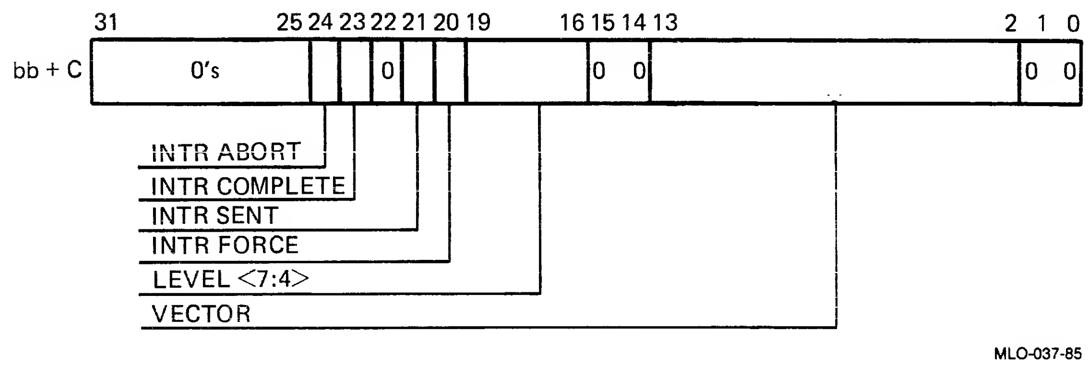


Figure 1-15 Error Interrupt Control Register (EINTRCSR)

Table 1-8 Error Interrupt Control Register (EINTRCSR) Bit Descriptions

Bit(s)	Name	Description
<31:25>	RESERVED	Not used (RO, ZEROS).
<24>	INTR ABORT (INTRAB)	Indicates an INTR command sent under the control of this register was aborted (R/W1C, DCLOC).
<23>	INTR COMPLETE (INTRC)	Indicates the vector for an error interrupt has been successfully transmitted or if an INTR command sent under the control of this register was aborted. Cleared when error interrupt request deasserted (R/W1C, DCLOC).
<22>	RESERVED	Not used (RO, ZERO).
<21>	INTR SENT	Indicates an INTR command under the control of this register has been sent. Cleared when error interrupt request deasserted (R/W1C, DCLOC, STOPC).
<20>	INTR FORCE	Forces an error interrupt request (R/W, DCLOC, STOPC).
<19:16>	LEVEL <7:4>	The interrupt level(s) for INTR commands sent under the control of this register. Also, the IDENT command level(s) this node (the NBIB) will respond to (R/W, DCLOC).
<15:14>	RESERVED	Not used (RO, ZEROS).
<13:2>	VECTOR	The vector sent by this node (the NBIB) during error interrupt sequences (R/W, DCLOC).
<1:0>	RESERVED	Not used (RO, ZEROS).

1.5.2.5 INTR Destination Register (INTRDES) -- The INTR destination register (Figure 1-16) controls which nodes are to be addressed by an INTR command. Register bits are defined in Table 1-9.

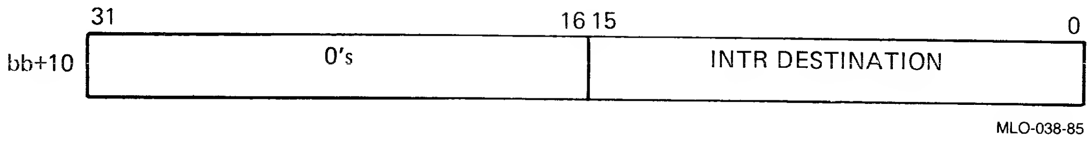


Figure 1-16 INTR Destination Register (INTRDES)

Table 1-9 INTR Destination Register (INTRDES) Bit Descriptions

Bit(s)	Name	Description
<31:16>	RESERVED	Not used (RO, ZEROS).
<15:0>	INTR DESTINATION (INTRDES)	Destination bits for INTR command. Transmitted on the VAXBI D lines during the command/address cycle of the INTR transaction. Each bit, when set, selects the corresponding VAXBI node. For example, bit <15> selects node 15; bit <0> selects node 0 (R/W, DCLOC).

1.5.2.6 IPINTR Mask Register (IPINTRMSK) -- The IPINTR mask register (Figure 1-17) controls which nodes are permitted to select this node (the NBIB) with an IPINTR command. Register bits are defined in Table 1-10.

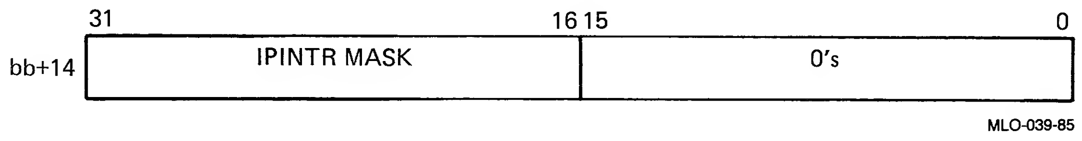


Figure 1-17 IPINTR Mask Register (IPINTRMSK)

Table 1-10 IPINTR Mask Register (IPINTRMSK) Bit Descriptions

Bit(s)	Name	Description
<31:16>	IPINTR MASK	Destination mask bits for IPINTR commands. Each bit, when set, allows the node (the NBIB) to respond to IPINTR commands from the corresponding VAXBI node minus 16. For example, bit <31> is the mask bit for node 15; bit <16> is the mask bit for node 0 (R/W, DCLOC).
<15:0>	RESERVED	Not used (RO, ZEROS).

1.5.2.7 IPINTR/STOP Destination Register (FIPSDES) -- The IPNTR/STOP destination register (Figure 1-18) controls which nodes are to be addressed by IPINTR and STOP commands. Register bits are defined in Table 1-11.

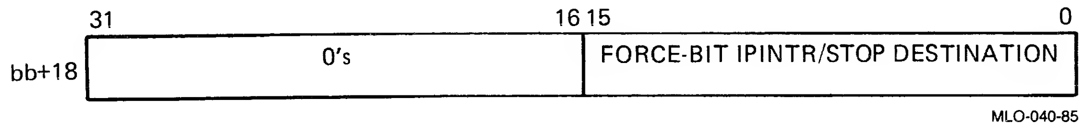


Figure 1-18 IPINTR/STOP Destination Register (FIPSDES)

Table 1-11 IPINTR/STOP Destination Register (FIPSDES) Bit Descriptions

Bit(s)	Name	Description
<31:16>	RESERVED	Not used (RO, ZEROS).
<15:0>	IPINTR/STOP DESTINATION	Destination bits for IPINTR and STOP commands. Transmitted on the VAXBI D lines during the command/address cycle of the IPINTR and STOP transactions. Each bit, when set, selects the corresponding VAXBI node. For example, bit <15> selects node 15; bit <0> selects node 0 (R/W, DCLOC).

1.5.2.8 IPINTR Source Register (IPINTRSRC) -- The IPINTR source register indicates the node that has sent an IPINTR command. Bit format is shown in Figure 1-19. Register bits are described in Table 1-12.

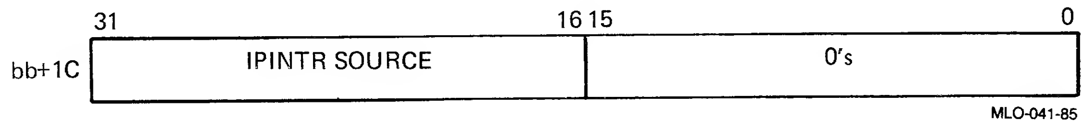


Figure 1-19 IPINTR Source Register (IPINTRSRC)

Table 1-12 IPINTR Source Register (IPINTRSRC) Bit Descriptions

Bit(s)	Name	Description
<31:15>	IPINTR SOURCE	Source bits for IPINTR commands received by this node (the NBIB). Each bit, when set, indicates the corresponding node minus 16 sent the command. For example, bit <31> indicates node 15 sent the IPINTR; bit <15> indicates node 0 sent the IPINTR. In order for a bit to be set, the appropriate bit in the IPINTR mask register must be set.
<15:0>	RESERVED	Not used (RO, ZEROS).

1.5.2.9 Starting and Ending Address Registers (SADR and EADR) -- The starting address register (Figure 1-20) defines the first location in a block of addresses that is to be recognized by the BIIC's slave port when responding to VAXBI read/write commands. The ending address register (Figure 1-21) defines the last location (plus one). The block of addresses may be in memory or I/O space, but the I/O space cannot include node space or multicast space. In the NBIB, a block of memory addresses is specified that defines the storage block in NMI memory used for DMA data transfers by the I/O devices connected to the VAXBI.

Register bits are described in Tables 1-13 and 1-14. The value in the starting and ending address fields specifies only the high-order twelve bits of address. The low-order eighteen bits are assumed to be all zeros by the BIIC logic. For example, if the starting address register contains 1C440000 (hex), and the ending address register contains 1D680000 (hex), the BIIC's slave port will respond to addresses 1C440000 through 1D67FFFF.

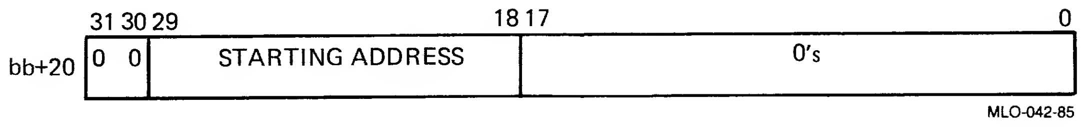


Figure 1-20 Starting Address Register (SADR)

Table 1-13 Starting Address Register (SADR) Bit Descriptions

Bit(s)	Name	Description
<31:30>	RESERVED	Not used (RO, ZEROS).
<29:18>	STARTING ADDRESS	Specifies the twelve high-order address bits of an address (the low-order address bits are assumed to be zeros), that is, the lowest address to be recognized by BIIC's slave port (R/W, DCLOC).
<17:0>	RESERVED	Not used (RO, ZEROS).

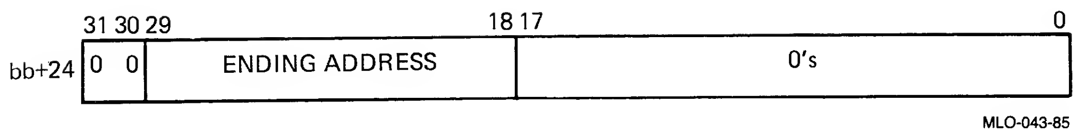


Figure 1-21 Ending Address Register (SADR)

Table 1-14 Ending Address Register (EADR) Bit Descriptions

Bit(s)	Name	Description
<31:30>	RESERVED	Not used. (RO, ZEROS)
<29:18>	STARTING ADDRESS	Specifies the twelve high-order address bits of an address (the low-order address bits are assumed to be zeros), that is, one greater than the highest address to be recognized by the BIIC's slave port (R/W, DCLOC).
<17:0>	RESERVED	Not used (RO, ZEROS).

1.5.2.10 BCI Control and Status Register (BCICSR) -- The BCI control/status register is shown in Figure 1-22. Bits are described in Table 1-15.

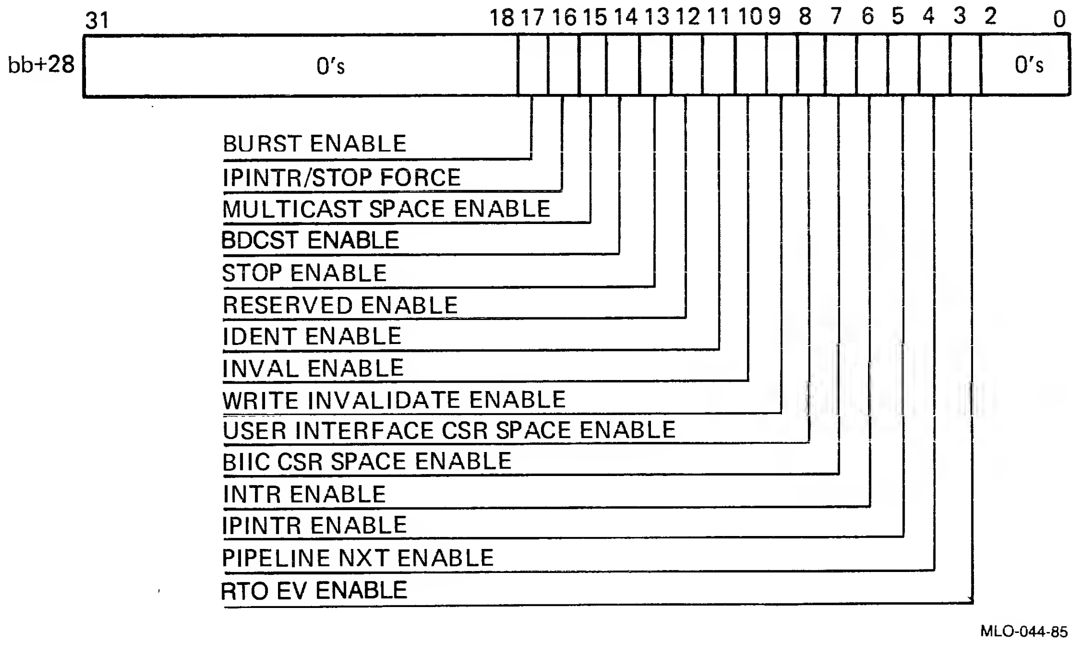


Figure 1-22 VAXBI Control/Status Register (BICSR)

Table 1-15 BCI Control/Status Register (BCICSR) Bit Descriptions

Bit(s)	Name	Description
<31:18>	RESERVED	Not used (RO, ZEROS).
<17>	BURST ENABLE (BURSTEN)	Causes BIIC to continuously assert VAXBI NO ARB following next successful arbitration by this node (the NBIB) (R/W, DCLOC).

Table 1-15 BCI Control/Status Register (BCICSR) Bit Descriptions (Cont)

Bit(s)	Name	Description
<16>	IPINTR/STOP FORCE	Causes BIIC to arbitrate for bus and transmit an IPINTR or STOP command. (The command transmitted depends on contents of the FIPSCMDS register (R/W, DCLOC).
<15>	MULTICAST SPACE ENABLE (MSEN)	Enables BIIC to assert BCI SEL and appropriate BCI SC<2:0> code following the receipt of a read/write command directed at multicast space (R/W, DCLOC).
<14>	BDCST ENABLE (BDCSTEN)	Enables BIIC to assert BCI SEL and appropriate BCI SC<2:0> code following the receipt of a BDCST command directed to this node (the NBIB) (R/W, DCLOC).
<13>	STOP ENABLE (STOPEN)	Enables BIIC to assert BCI SEL and appropriate BCI SC<2:0> code following the receipt of a STOP command directed to this node (the NBIB) (R/W, DCLOC).
<12>	RESERVED ENABLE (RESEN)	Enables BIIC to assert BCI SEL and appropriate BCI SC<2:0> code following the receipt of a RESERVED command directed to this node (the NBIB) (R/W, DCLOC).
<11>	IDENT ENABLE (RESEN)	Enables BIIC to assert BCI SEL and appropriate BCI SC<2:0> code following the receipt of an IDENT command directed to this node (the NBIB). When bit is cleared, BIIC still participates in transaction even though SEL and SC<2:0> are not asserted (R/W, DCLOC).
<10>	INVAL ENABLE (INVALEN)	Enables BIIC to assert BCI SEL and appropriate BCI SC<2:0> code following the receipt of an INVAL command directed to this node (the NBIB) (R/W, DCLOC).

Table 1-15 BCI Control/Status Register (BCICSR) Bit Descriptions (Cont)

Bit(s)	Name	Description
<9>	WRITE INVALIDATE ENABLE (WINVALEN)	Enables BIIC to assert BCI SEL and appropriate BCI SC<2:0> code following the receipt of a write command whose address falls outside the range specified by the starting and ending address registers and it is a memory address (R/W, DCLOC).
<8>	USER INTERFACE CSR SPACE ENABLE (UCSREN)	Enables BIIC to assert BCI SEL and appropriate BCI SC<2:0> code following the receipt of a read/write command directed at user interface CSR space (R/W, DCLOC).
<7>	BIIC CSR SPACE ENABLE (UCSREN)	Enables BIIC to assert BCI SEL and appropriate BCI SC<2:0> code following the receipt of a read/write command directed at the BIIC's CSR space. When bit is cleared, BIIC still participates in transaction even though SEL and SC<2:0> are not asserted (R/W, DCLOC).
<6>	INTR ENABLE (INTREN)	Enables BIIC to assert BCI SEL and appropriate BCI SC<2:0> code following the receipt of an INTR command directed to this node (the NBIB) (R/W, DCLOC).
<5>	IPINTR ENABLE (IPINTREN)	Enables BIIC to assert BCI SEL and appropriate BCI SC<2:0> code following the receipt of an IPINTR command from a node included in the IPINTR mask register (R/W, DCLOC).
<4>	PIPELINE NXT ENABLE (PNXTEN)	Causes the BIIC to assert BCI NXT for an extra cycle during write and BDCST transactions. Facilitates the use of FIFO pointers for some nodes (not the NBIB) (R/W, DCLOC).
<3>	RTO EV ENABLE (RTOEVN)	Enables the BIIC to assert a RETRY TIMEOUT code on the BCI EV<4:0> lines after a retry timeout (R/W, DCLOC).
<2:0>	RESERVED	Not used (RO, ZEROS).

1.5.2.11 Write Status Register (WSTAT) -- The write status register (Figure 1-23 and Table 1-16) indicates which general-purpose registers have been written. A status bit is set only if good parity is received with the write data. (The general-purpose registers are not used in the NBIB node.)

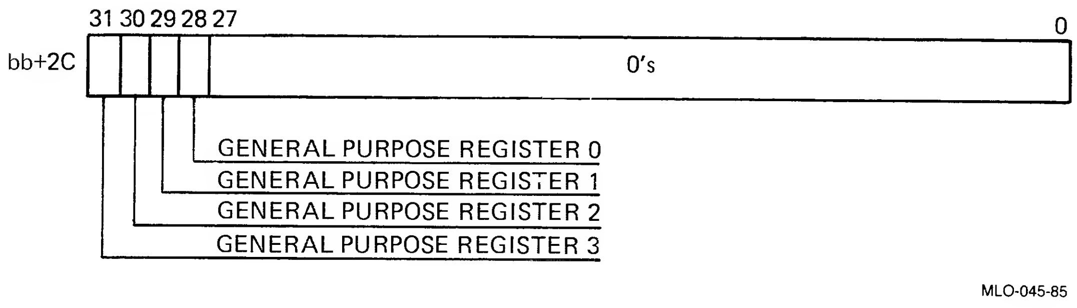

31 30 29 28 27
0
bb+2C
0's
GENERAL PURPOSE REGISTER 0
GENERAL PURPOSE REGISTER 1
GENERAL PURPOSE REGISTER 2
GENERAL PURPOSE REGISTER 3
MLO-045-85

Figure 1-23 Write Status Register (WSTAT)

Table 1-16 Write Status Register (WSTAT) Bit Descriptions

Bit(s)	Name	Description
<31>	GENERAL PURPOSE REGISTER 3 (GPR3)	General-purpose register 3 written (R/W1C, DCLOC)
<30>	GENERAL PURPOSE REGISTER 2 (GPR2)	General-purpose register 2 written (R/W1C, DCLOC)
<29>	GENERAL PURPOSE REGISTER 1 (GPR3)	General-purpose register 1 written (R/W1C, DCLOC)
<28>	GENERAL PURPOSE REGISTER 0 (GPR0)	General-purpose register 0 written (R/W1C, DCLOC)
<27:0>	RESERVED	Not used (RO, ZERO)

1.5.2.12 Force IPINTR/STOP Command Register (FIPSCMD) -- The force IPINTR/STOP command register (Figure 1-24 and Table 1-17) contains the code for the command initiated by the BIIC when the IPINTR/STOP FORCE bit is set in the BCI control/status register (BCICSR). Only an IPINTR or STOP command code should be loaded in the force IPINTR/STOP command register.

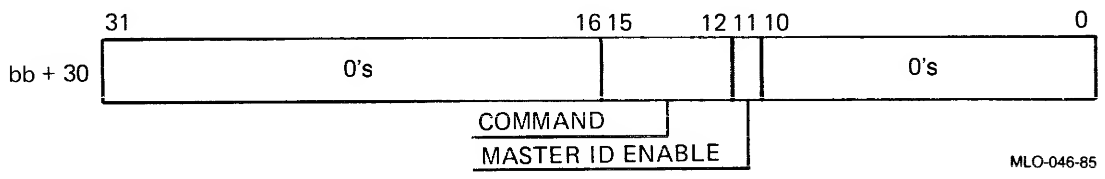


Figure 1-24 Force IPINTR/STOP Command Register (FIPSCMD)

Table 1-17 Force IPINTR/STOP Command Register (FIPSCMD) Bit Descriptions

Bit(s)	Name	Description
<31:16>	RESERVED	Not used (RO, ZEROS).
<15:12>	COMMAND (CMD)	IPINTR OR STOP command code for VAXBI transaction initiated by BIIC when the IPINTR/STOP FORCE bit is set in the BCICSR (R/W, DCLOS).
<11>	MASTER ID ENABLE (MIDEN)	Causes the node's (the NBIB's) ID to be transmitted on the VAXBI D lines during the transaction initiated by the IPINTR/STOP FORCE bit (in the BCICSR). Must be set when an IPINTR command is in the command field.
<10:0>	RESERVED	Not used (RO, ZEROS).

1.5.2.13 User Interrupt Control Register (UINTRCSR) -- The user interrupt control register controls the operation of interrupts initiated by the user interface (the NBIB logic). Register bit format is shown in Figure 1-25. Bit definitions are given in Table 1-18.

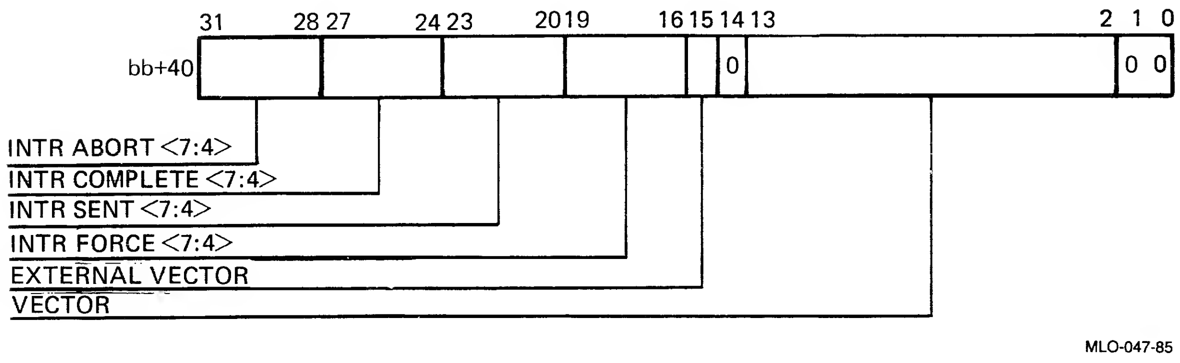


Figure 1-25 User Interrupt Control Register (UINTRCSR)

Table 1-18 User Interrupt Control Register (UINTRCSR) Bit Descriptions

Bit(s)	Name	Description
<31:28>	INTR ABORT (INTRAB) <7:4>	Indicate an INTR command sent under the control of this register was aborted. The bit(s) set indicate the INTR level(s) BR<7:4> (R/W1C, DCLOC).
<27:24>	INTR COMPLETE (INTRC) <7:4>	Indicate the vector for an interrupt has been successfully transmitted or if an INTR command sent under the control of this register was aborted. The bit(s) set indicate the INTR level(s) BR<7:4>. A bit is cleared if the corresponding error interrupt request is deasserted (R/W1C, DCLOC).
<23:20>	INTR SENT (SENT) <7:4>	Indicate an INTR command sent under the control of this register has been sent. The bit(s) set indicate the INTR level(s) BR<7:4>. A bit is cleared by an IDENT servicing the corresponding interrupt level (R/W1C, DCLOC, STOPC).
<19:16>	INTR FORCE (FORCE) <7:4>	Forces an interrupt request. The bit(s) set determines the interrupt request level(s) BR<7:4> (R/W, DCLOC, STOPC).
<15>	EXTERNAL VECTOR (EX VECTOR)	Causes BIIC to send vector supplied by user interface on BCI D lines during user interface interrupt sequences (R/W, DCLOC).
<14>	RESERVED	Not used (RO, ZEROS).
<13:2>	VECTOR	The vector sent by this node (the NBIB) during user interface interrupt sequences unless the EX VECTOR control bit is set.
<1:0>	RESERVED	Not used (RO, ZEROS).

1.5.2.14 General Purpose Registers (GPR <3:0>) -- The BIIC's general-purpose registers (Figure 1-26) are not used by the NBIB. However, the registers may be accessed and they are cleared when the NBIB is initialized (R/W, DCLOC).

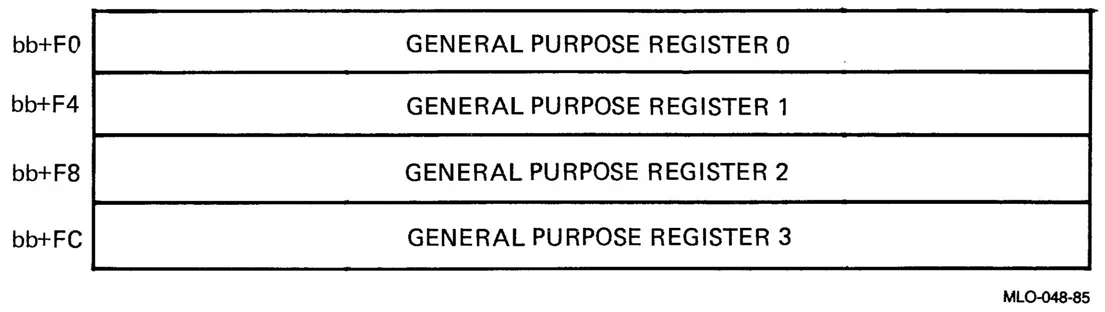

bb+F0
GENERAL PURPOSE REGISTER 0
bb+F4
GENERAL PURPOSE REGISTER 1
bb+F8
GENERAL PURPOSE REGISTER 2
bb+FC
GENERAL PURPOSE REGISTER 3
MLO-048-85

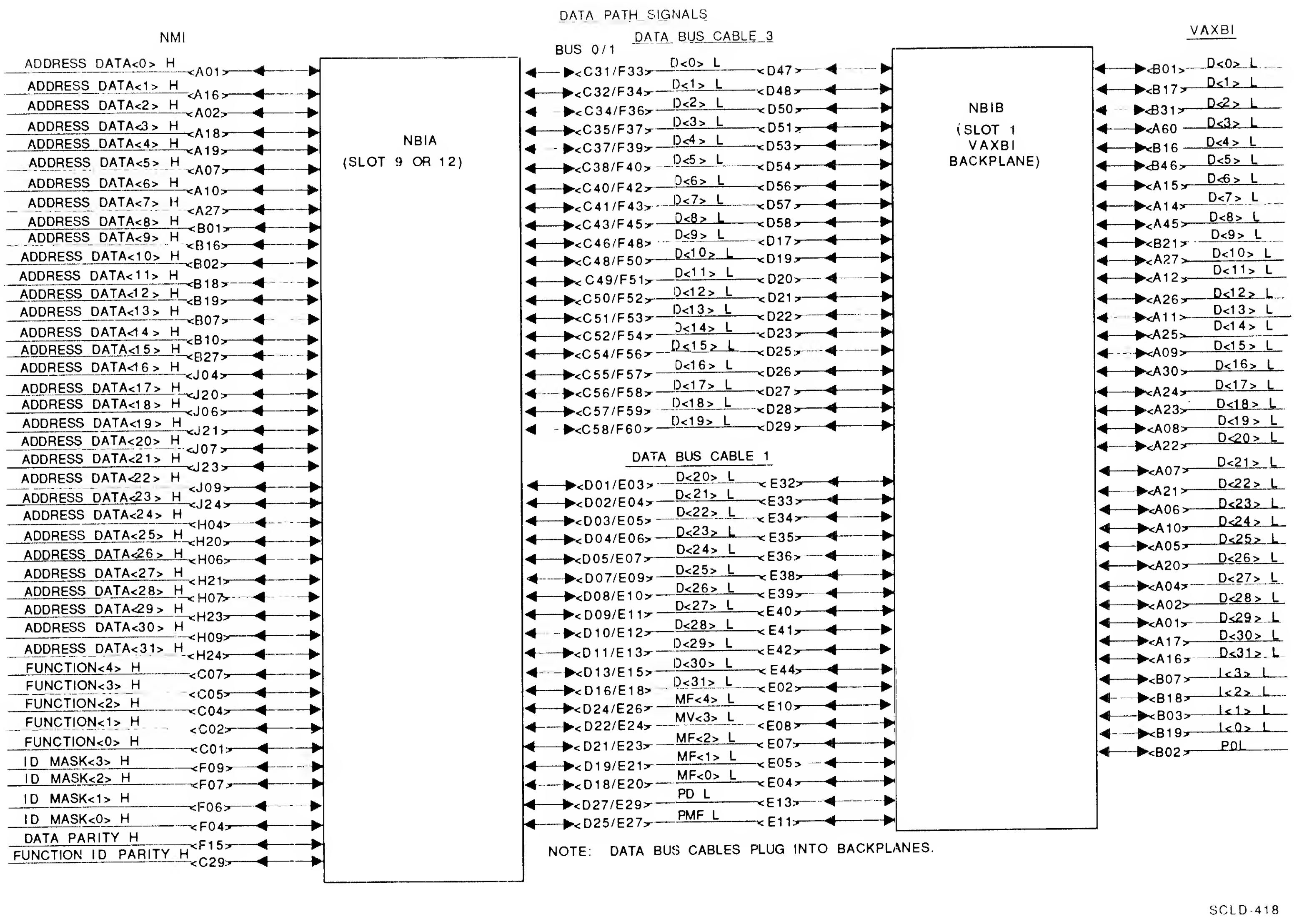
Figure 1-26 General-Purpose Registers (GPR <3:0>)

CHAPTER 2
INTERFACE DESCRIPTIONS

2.1 NMI

The NMI signal lines connecting to the NBIA, and the VAXBI signal lines connecting to the NBIB, are shown in Figure 2-1. Also shown are the data bus signal lines connecting the NBIA and NBIB. Backplane pin assignments are indicated.

The NMI is the main system interconnect. It is a synchronous bus that interconnects the CPU(s), memory, and any I/O adapter modules such as the NBIA(s). A detailed description of the NMI is given in System Bus Summary (Section 2 of this manual). A basic description follows.



NOTE: DATA BUS CABLES PLUG INTO BACKPLANES.

Figure 2-1 NBIA and NBIB Input/Output Signals (Sheet 1 of 2)

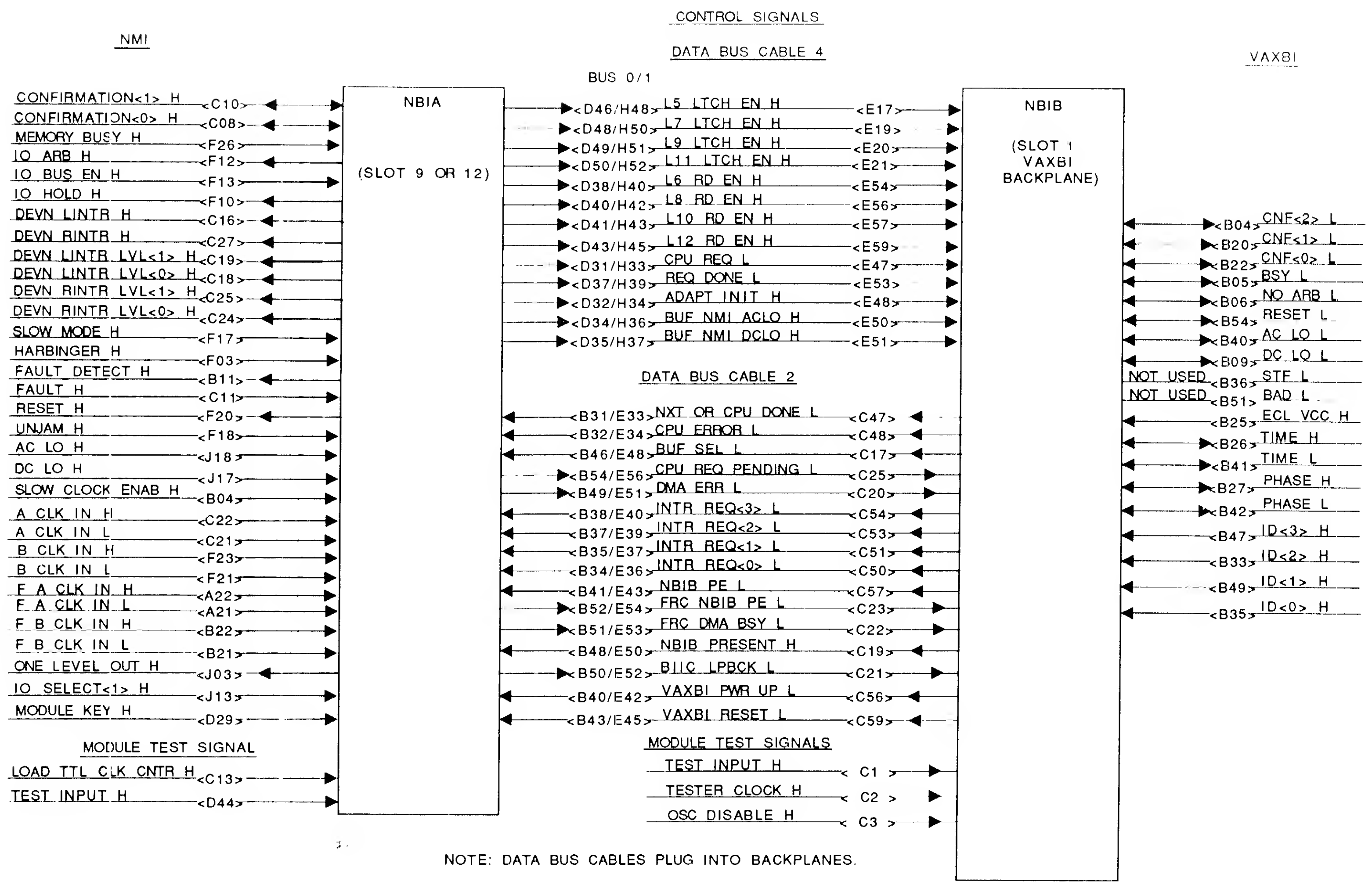

NMI
CONTROL SIGNALS
DATA BUS CABLE 4
BUS 0/1
VAXBI
NBIA
(SLOT 9 OR 12)
NBIB
(SLOT 1 VAXBI BACKPLANE)
CONFIRMATION<1> H <C10>
CONFIRMATION<0> H <C08>
MEMORY BUSY H <F26>
IO ARB H <F12>
IO BUS EN H <F13>
IO HOLD H <F10>
DEVN LINTR H <C16>
DEVN RINTR H <C27>
DEVN LINTR LVL<1> H <C19>
DEVN LINTR LVL<0> H <C18>
DEVN RINTR LVL<1> H <C25>
DEVN RINTR LVL<0> H <C24>
SLOW MODE H <F17>
HARBINGER H <F03>
FAULT DETECT H <B11>
FAULT H <C11>
RESET H <F20>
UNJAM H <F18>
AC LO H <J18>
DC LO H <J17>
SLOW CLOCK ENAB H <B04>
A CLK IN H <C22>
A CLK IN L <C21>
B CLK IN H <F23>
B CLK IN L <F21>
F A CLK IN H <A22>
F A CLK IN L <A21>
F B CLK IN H <B22>
F B CLK IN L <B21>
ONE LEVEL OUT H <J03>
IO SELECT<1> H <J13>
MODULE KEY H <D29>
MODULE TEST SIGNAL
LOAD TTL CLK CNTR H <C13>
TEST INPUT H <D44>
<D46/H48> L5 LTCH EN H <E17>
<D48/H50> L7 LTCH EN H <E19>
<D49/H51> L9 LTCH EN H <E20>
<D50/H52> L11 LTCH EN H <E21>
<D38/H40> L6 RD EN H <E54>
<D40/H42> L8 RD EN H <E56>
<D41/H43> L10 RD EN H <E57>
<D43/H45> L12 RD EN H <E59>
<D31/H33> CPU REQ L <E47>
<D37/H39> REQ DONE L <E53>
<D32/H34> ADAPT INIT H <E48>
<D34/H36> BUF NMI ACLO H <E50>
<D35/H37> BUF NMI DCLO H <E51>
DATA BUS CABLE 2
<B31/E33> NXT OR CPU DONE L <C47>
<B32/E34> CPU ERROR L <C48>
<B46/E48> BUF SEL L <C17>
<B54/E56> CPU REQ PENDING L <C25>
<B49/E51> DMA ERR L <C20>
<B38/E40> INTR REQ<3> L <C54>
<B37/E39> INTR REQ<2> L <C53>
<B35/E37> INTR REQ<1> L <C51>
<B34/E36> INTR REQ<0> L <C50>
<B41/E43> NBIB PE L <C57>
<B52/E54> FRC NBIB PE L <C23>
<B51/E53> FRC DMA BSY L <C22>
<B48/E50> NBIB PRESENT H <C19>
<B50/E52> BIIC LPBCK L <C21>
<B40/E42> VAXBI PWR UP L <C56>
<B43/E45> VAXBI RESET L <C59>
MODULE TEST SIGNALS
TEST INPUT H < C1 >
TESTER CLOCK H < C2 >
OSC DISABLE H < C3 >
<B04> CNF<2> L
<B20> CNF<1> L
<B22> CNF<0> L
<B05> BSY L
<B06> NO ARB L
<B54> RESET L
<B40> AC LO L
<B09> DC LO L
NOT USED <B36> STF L
NOT USED <B51> BAD L
<B25> ECL VCC H
<B26> TIME H
<B41> TIME L
<B27> PHASE H
<B42> PHASE L
<B47> ID<3> H
<B33> ID<2> H
<B49> ID<1> H
<B35> ID<0> H
NOTE: DATA BUS CABLES PLUG INTO BACKPLANES.
SCLD-419

Figure 2-1 NBIA and NBIB Input/Output Signals (Sheet 2 of 2)

2.1.1 NMI Signals

NMI signal lines connecting to an NBIA are defined in Table 2-1. All the signals are ECL except for ACLO and DCLO, which are FET-driven and received.

2.1.2 Basic Timing

Generally, NMI signals are asserted and deasserted at the beginning of a bus cycle, which is phase B of the system clock. Also, bus signals are received and latched using phase A of the system clock. Refer to Figure 2-2.

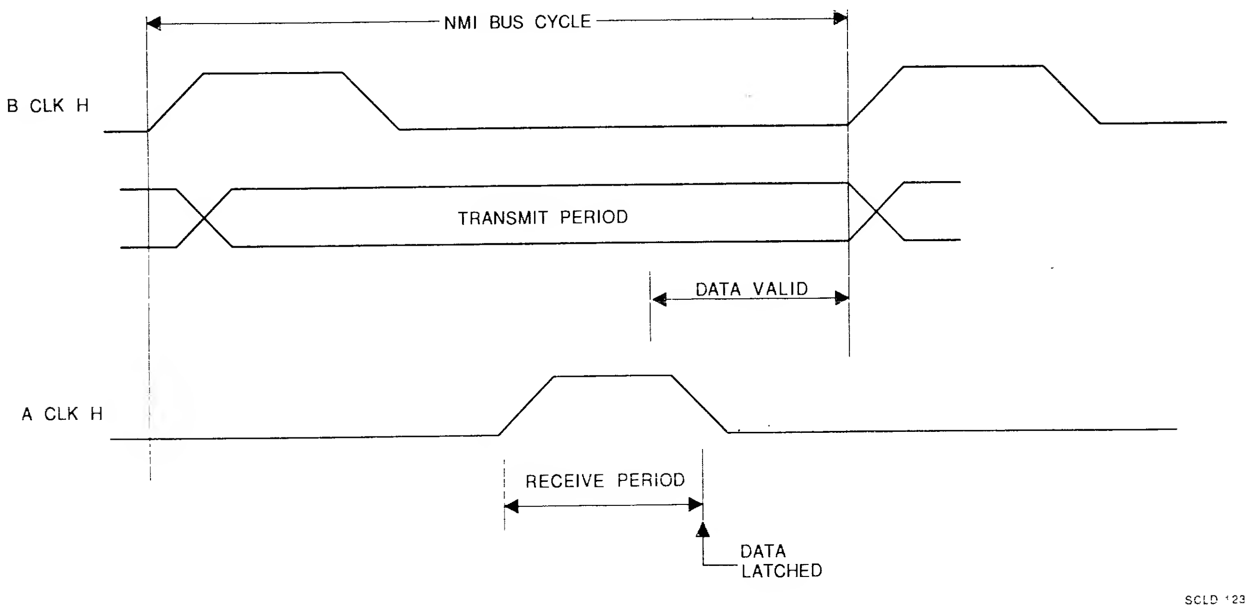


Figure 2-2 Basic NMI Timing

Table 2-1 NMI Signals Connecting to NBIA

Signal(s)	Number	Description
ADDRESS DATA <31:00>	32	Multiplexed address/data lines. Transfer 30-bit address during command/address cycles. Transfer longword of read/write data during data cycles.
FUNCTION <4:0>	5	Specify command during command/-address cycles and data type during data cycles.

FUNCTION <4:0> (Hex)	Command/Data Type
10	Read Longword
12	Read Octaword
13	Read Hexword
14	Read Longword (Interlocked)
16	Read Octaword (Interlocked)
17	Read Hexword (Interlocked)
1B	Write Longword
1F	Write Octaword
18	Write Longword Masked
19	Write Quadword Masked
1A	Write Octaword Masked
1C	Write Longword Unlock Masked
1D	Write Quadword Unlock Masked
1E	Write Octaword Unlock Masked
00	No Op
04	Memory Pause (Not Used)
0A	Return Read Data (Good Data)
0E	Return Read Data (Bad Data)
08	Read Continue (Good Data)
0C	Read Continue (Bad Data)
09	Write Data

Table 2-1 NMI Signals Connecting to NBIA (Cont)

Signal(s)	Number	Description
ID MASK<3:0> H	4	Specify ID of commander during command/address cycles and return read data cycles. No ID is required for the memory nexus as it is never the commander during a bus transaction.

ID	Nexus
0100	CPU 0 (Primary CPU)
0101	
* 0110	CPU 1 (Attached CPU)
* 0111	
* 1000	NBIA 0 (VAXBI0)
* 1001	
* 1010	NBIA 0 (VAXBI1)
* 1011	
1100	NBIA 1 (VAXBI0)
1101	
1110	NBIA 1 (VAXBI1)
1111	

* Dual-CPU backplanes only

Also, during NMI write data cycles of masked write transactions, specify the bytes in the longword (four bytes) of write data that are to be written.

MASK BIT 3 2 1 0	
X X X 1	Write byte 0
X X 1 X	Write byte 1
X 1 X X	Write byte 2
1 X X X	Write byte 3

Signal(s)	Number	Description
DATA PARITY H	1	Transfers even parity bit for the 32 ADDRESS DATA lines during NMI command/address and data cycles.

Table 2-1 NMI Signals Connecting to NBIA (Cont)

Signal(s)	Number	Description
FUNCTION ID PARITY H	1	Transfers even parity bit for the 5 FUNCTION lines and 4 ID MASK lines during NMI command/address and data cycles.
CONFIRMATION <1:0> H	2	Specify response (by responder) to function sent by commander.
		CONFIRMATION <1:0> — Response
		00 — No response (NOACK)
		01 — Command/address acknowledged (ACK)
		10 — Interlocked
		11 — Busy
MEMORY BUSY H	1	Asserted by memory nexus when its command/data buffers are full. Causes current transmitter to abort transaction and rearbitrate for the bus when asserted during command/address cycle of memory read, or during command/address or first data cycle of memory write.
IO ARB H	1	Asserted by NBIA to request use of the bus.
IO BUS EN H	1	Asserted by bus arbitrator to grant NBIA use of the bus.
IO HOLD H	1	Asserted by NBIA when it requires another (more than one) bus cycle after getting use of the bus.
DEVN LINTR H	1	Asserted by NBIA to interrupt the CPU (the left CPU in dual-processor systems).
DEVN RINTR H	1	Asserted by NBIA to interrupt the right CPU in dual-processor systems.
DEVN LINTR LVL<1:0> H	2	Asserted by NBIA to specify the interrupt request level to the CPU (the left CPU in dual-processor systems).

Table 2-1 NMI Signals Connecting to NBIA (Cont)

Signal(s)	Number	Description
DEVN RINTR LVL<1:0> H	2	Asserted by NBIA to specify the interrupt request level to the right CPU in dual-processor systems. DEV (L/R)INTR LVL <1:0> — Interrupt Request Level 00 — BR4 01 — BR5 10 — BR6 11 — BR7
FAULT DETECT H	1	Asserted by NBIA when it has detected a fault. Signal is ORed with FAULT DETECT lines from other nexus in memory nexus.
FAULT H	1	Asserted by fault handling logic in memory nexus to signal a FAULT DETECT line has been asserted.
UNJAM H	1	Initializes all nexus without clearing status indicators. Also, causes NBIB(s) connected to NBIA to generate an ACLO/DCLO sequence on the VAXBI. Asserted by clock module in response to a console command.
RESET H	1	Asserted by NBIA if VAXBI RESET is asserted by a VAXBI device connected to the NBIB. Causes console to stop the CPU(s) and then bootstrap the system (a cold start). The bootstrap does not occur until RESET is negated.
SLOW CLOCK ENABLE H	1	Asserted by clock module and used to increment timeout counters in all nexus.
SLOW MODE H	1	Early-warning signal asserted by clock module to indicate that the normal system clocks (A CLK and B CLK) are about to start or stop. Asserted a minimum of 4 microseconds before HARBINGER.

Table 2-1 NMI Signals Connecting to NBIA (Cont)

Signal(s)	Number	Description
HARBINGER H	1	Asserted by clock module during last B PHASE clock when normal system clocks are stopped. Used as clock blocking signal by nexus using free-running clocks(F A CLK and F B CLK) to simulate stopped-clock behavior. Not used by NBIA.
AC LO H	1	FET-driven and received power-loss warning signal asserted by system power supply when ac power is below specified limits.
DC LO H	1	FET-driven and received power-loss warning signal asserted by system power supply when dc power is below specified limits.
IO SELECT<1> H	1	Backplane-generated (hard-wired) signal that determines I/O address space for NBIA. A logical one is generated by backplane wiring that connects IO SEL input to the NBIA's ONE LEVEL output. (No connection is equivalent to a logical zero.)
ONE LEVEL OUT H	1	Logic level output (logical one) asserted by NBIA. Used to generate IO SEL input signal on the backplane.

IO SELECT <1>	NBIA	NMI Base Address (Hex)
0	0	2000 0000
1	1	2400 0000

Table 2-1 NMI Signals Connecting to NBIA (Cont)

Signal(s)	Number	Description
A CLK IN H/L	2	Phase A of normal system clock. Generated by clock module.
B CLK IN H/L	2	Phase B of normal system clock. Generated by clock module. An NMI bus cycle is defined as the period between the leading edge of one B CLK and the next.
F A CLK IN H/L	2	Phase A of free-running system clock. Not used by NBIA.
F B CLK IN H/L	2	Phase B of free-running system clock. Not used by NBIA.
MODULE KEY H	1	Module keying loop. No connection within NBIA module. This line is grounded if the wrong module type is plugged into the NBIA slot.

2.1.3 NMI Address Space

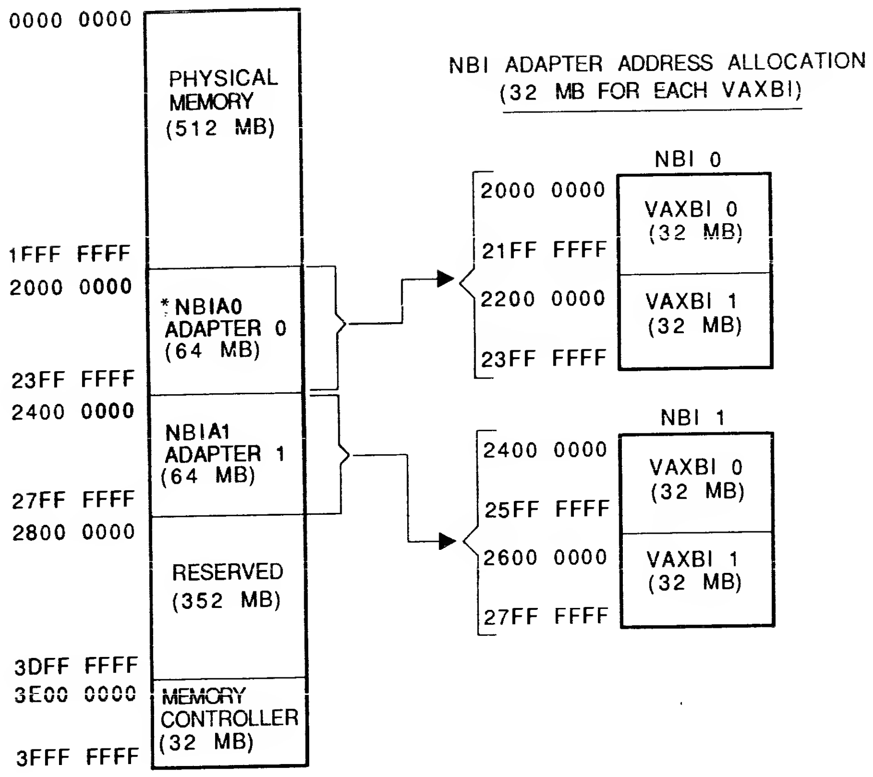
NMI address space is 1 Gbyte. (Only 30 of the 32 multiplexed NMI ADDRESS/DATA lines are used for addressing purposes.) As shown in Figure 2-3, one half of the address space is allocated to physical memory. The other half is I/O address space. Within the I/O space, a 64-Mbyte block is allocated to each NBIA (32 Mbytes for each VAXBI) in the system. When up to two NBIAs can be installed in a system (dual-CPU backplane), the first 64-Mbyte block is allocated to NBIA0 and the second to NBIA1. (The single NBIA in a single-CPU backplane is NBIA1.)

2.1.4 NMI Read/Write Transactions

Before an NMI nexus (an NBIA, a CPU, or the memory) can transfer information on the NMI, it must first request and then be granted use of the bus. (Bus arbitration is discussed in Section 2.1.5.)

A write transaction consists of a single transfer. Once it has been granted the bus, the commander (the nexus originating the transaction) transmits the command/address information and one or more longwords of write data during consecutive bus cycles. This ends the transaction. A read transaction, on the other hand, requires more than one transfer. First, after it has been granted the bus, the commander transmits the command/address. Then, the responder (the nexus addressed by the commander) must request and be granted the bus in order to return the read data to the commander. For read hexword transactions, two return read data transfers (each transferring one octaword) are made.

Timing for NMI read/write transactions is shown in Figures 2-4 and 2-5. The NBIA originates read (longword/octaword) transactions and write (longword/quadword/octaword) transactions to access memory during DMA data transfers by a VAXBI device. (Read hexword transactions, although supported on the NMI, are not originated by the NBIA.) Also, the NBIA responds to read/write (longword) transactions originated by the CPU when a VAXBI device register, or NBIA CSR, is addressed.



*RESERVED ADDRESS SPACE IN SYSTEMS THAT CAN HAVE ONLY ONR NBIA INSTALLED (SYSTEMS HAVING SINGLE-CPU BACKPLANES). IN THESE SYSTEMS, THE SINGLE NBIA IS NBIA1.

SCLD-128

Figure 2-3 NMI Address Space

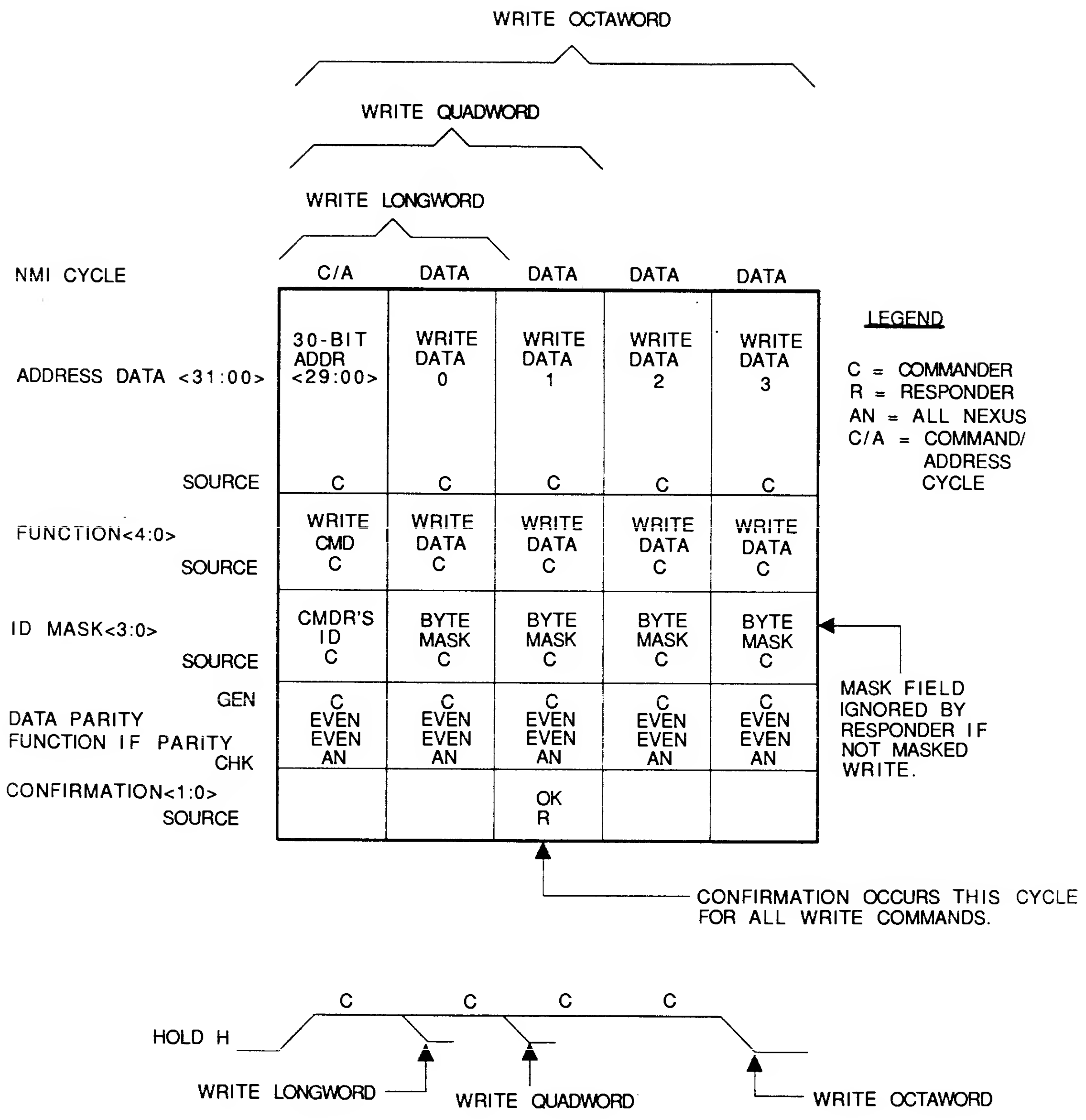
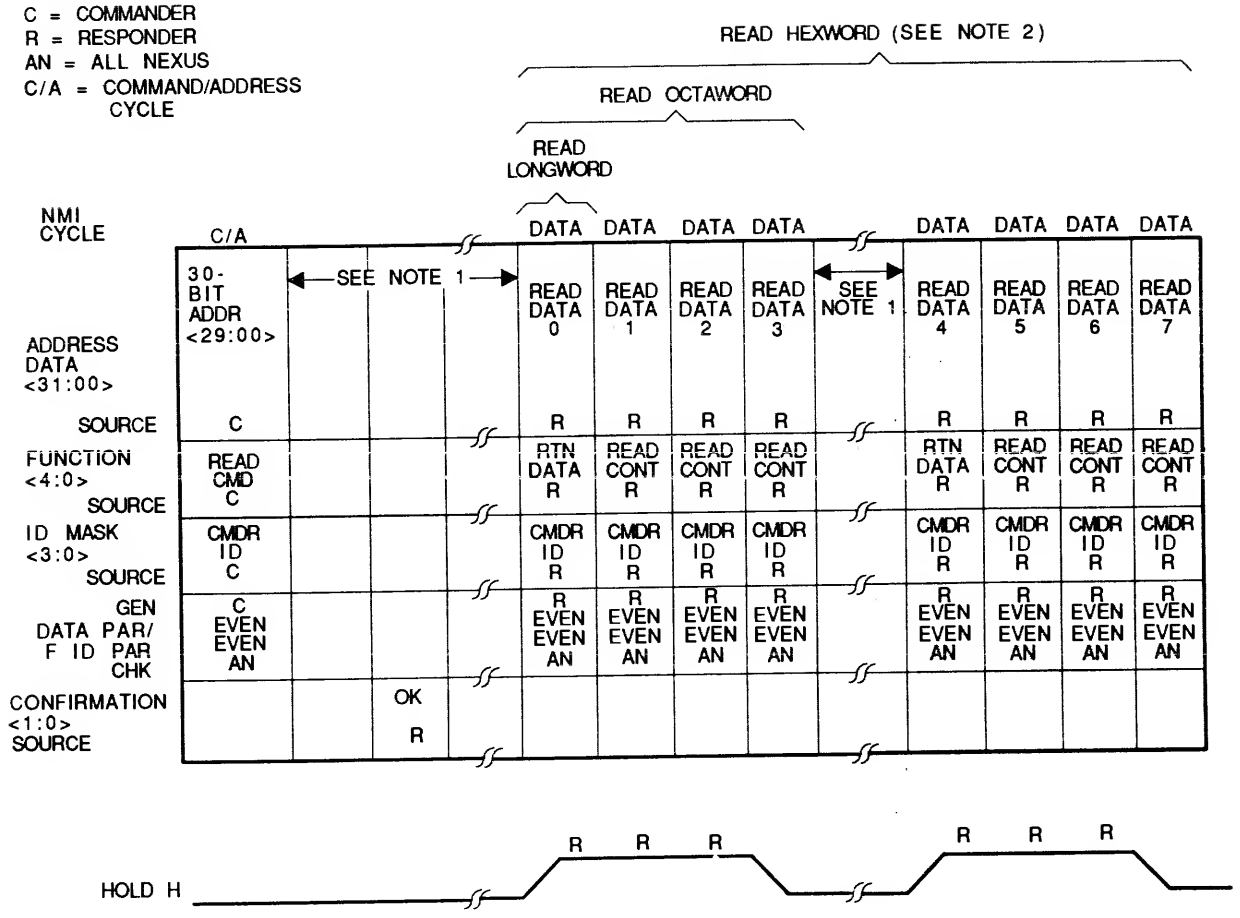


Figure 2-4 NMI Write Transaction

LEGEND

C = COMMANDER
R = RESPONDER
AN = ALL NEXUS
C/A = COMMAND/ADDRESS CYCLE



NOTE 1: NMI AVAILABLE FOR OTHER TRANSACTIONS.

NOTE 2: RESPONDER (MEMORY) RELEASES NMI AFTER TRANSFER OF FIRST OCTAWORD IN THE READ HEXWORD TRANSACTION SHOW. RESPONDER (MEMORY) MAY ALSO HOLD BUS AND TRANSFER ALL THE DATA (EIGHT SUCCESSIVE DATA CYCLES) IF SECOND OCTAWORD IS READY FOR TRANSMISSION.

SCLD-132

Figure 2-5 NMI Read Transaction

During the command/address cycle of a transaction, the commander transmits the address on the ADDRESS DATA lines, the read/write function (read longword, write octaword, etc.) on the FUNCTION lines, and the commander's ID on the ID MASK lines. If the responder accepts the command/address, it asserts an ACK (acknowledge) response on the CONFIRMATION lines. A BUSY response is asserted if a responder (the memory, for example) is not ready to accept the command/address at this time. The memory can also assert an INTERLOCKED response. The memory is hardware interlocked by a read interlocked transaction. Although it executes the first interlocked transaction normally, any read interlocked transactions that follow are not accepted until the interlock has been removed by a write unlock transaction.

During the write data cycle(s) immediately following a write transaction's command/address cycle, the commander transmits the write data on the ADDRESS DATA lines, a code identifying the data (as write data) on the FUNCTION lines, and a byte mask on the ID MASK lines if the transaction is a masked write. The byte mask indicates which bytes in the longword of write data are to be written.

During read data cycles, the responder transmits the read data on the ADDRESS DATA lines, a code identifying the data as read data on the FUNCTION lines, and the (saved) commander's ID on the ID MASK lines. The FUNCTION line code indicates if the read data is the first longword in the transfer and if the data is good or bad. That is, a RETURN READ DATA (GOOD or BAD) code is returned with the first longword. A READ CONTINUE (GOOD or BAD) code is returned with all other longwords.

Two separate even parity bits are transmitted by the commander during command/address and write data cycles, and by the responder during read data cycles. One parity bit, which is generated for the information on the ADDRESS DATA lines, is transmitted on the DATA PARITY line. The other parity bit, which is generated for the information on the FUNCTION and ID MASK lines, is transmitted on the FUNCTION ID PARITY line. The two parity bits are checked by all nexus. If bad (odd) parity is detected, it causes the NMI FAULT line to be asserted. Parity and other NMI errors are discussed in Section 2.1.7.

2.1.5 NMI Arbitration/Memory Busy

The bus arbitrator for the NMI is in the CPU (the left CPU in dual-processor systems). Each NMI nexus has a bus arbitration (ARB) line, a BUS EN line, and a HOLD line connecting to the arbitrator. Basic arbitration line timing is shown in Figure 2-6.

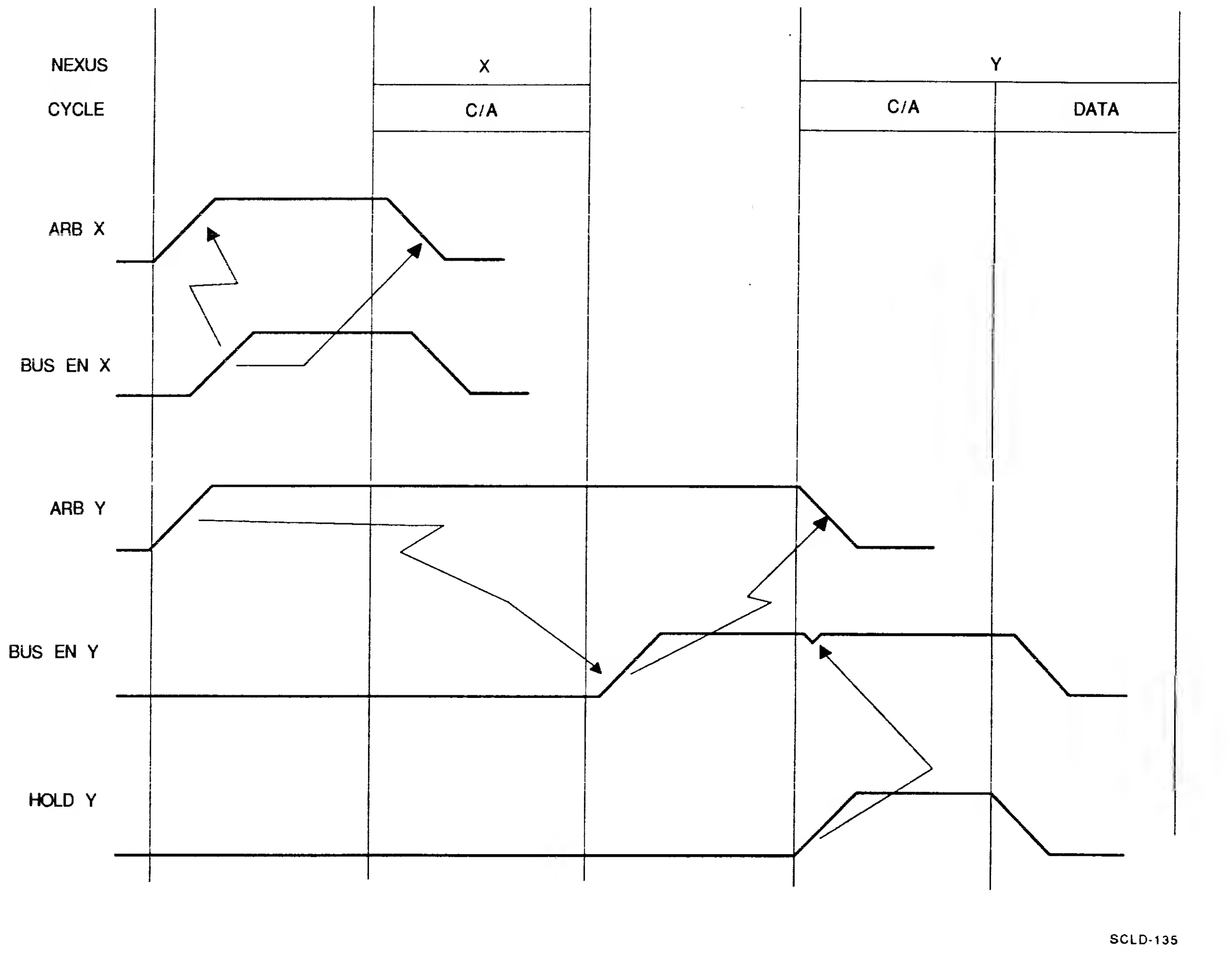


Figure 2-6 Basic NMI Arbitration Line Timing

To request use of the bus, each nexus must assert its ARB request line. The request line is asserted at the beginning of a bus cycle. More than one nexus may request the bus at a time and the bus arbitrator resolves request priority and asserts the BUS EN line for the winning nexus. The BUS EN line is asserted immediately if the bus is not currently in use. If the bus is in use, the BUS EN line is not asserted until the bus is free. This ensures a dead cycle between transfers on the NMI, which is necessary due to electrical constraints. When a nexus receives its BUS EN signal, it negates its ARB request line at the beginning of the next bus cycle.

If a nexus wins the bus and requires more than one bus cycle for a transfer, it must assert its HOLD line to keep the bus. The HOLD line is asserted at the beginning of the first bus cycle in the transfer. It is negated at the beginning of the last bus cycle in the transfer. In the bus arbitrator, the asserted HOLD line disables any further bus arbitration. It also causes the BUS EN line to the nexus holding the bus to remain asserted.

The memory nexus has the highest priority when requesting the bus. The CPUs and NBIAs in the system have the lowest priority and share the bus on an alternating basis. If both a CPU and an NBIA are requesting the bus, and if a CPU rather than an NBIA won the bus last, then the NBIA wins the bus. Conversely, if a CPU rather than an NBIA had the bus last, the CPU wins. Furthermore, when two NBIAs (or two CPUs) in the system are requesting the bus and an NBIA (or CPU) wins, it is the one that did not use the bus last. Typical arbitration line timing in a system is shown in Figure 2-7.

In addition to having the highest priority, the memory nexus may assert MEMORY BUSY. This signal, by means of the MEMORY BUSY ARB line (a copy of MEMORY BUSY connecting to the bus arbitrator), prevents the arbitrator from granting CPU or NBIA bus requests. Bus requests by the memory itself can still be granted. The memory asserts MEMORY BUSY when its command/data buffers are full, thus preventing the other nexus from using the bus until it can empty the buffers and respond to a command/address.

If MEMORY BUSY is asserted during the command/address cycle of a memory read/write transaction (refer to Figure 2-8), it indicates that the command/address will not be accepted by the memory. In this sense, MEMORY BUSY acts as an early-warning status line in addition to its function of temporarily preventing further memory read/write transactions on the bus. That is, the nexus transmitting the command/address does not have to wait for the BUSY response on the CONFIRMATION lines before aborting and then retrying the transaction. This allows the nexus to retry before another (lower priority) request gets the bus should memory become available in the shortest possible time. (MEMORY BUSY is asserted for only two bus cycles in some cases.)

When MEMORY BUSY is asserted after the command/address cycle and during the first data cycle of a memory write transaction, it also indicates the command will not be accepted by the memory. Again, the transaction can be retried without having to wait for the BUSY response on the CONFIRMATION lines. If MEMORY BUSY is asserted after the first data cycle (write quadword or write octaword transactions), the command and data will be accepted by the memory unless otherwise indicated by the response returned on the CONFIRMATION lines.

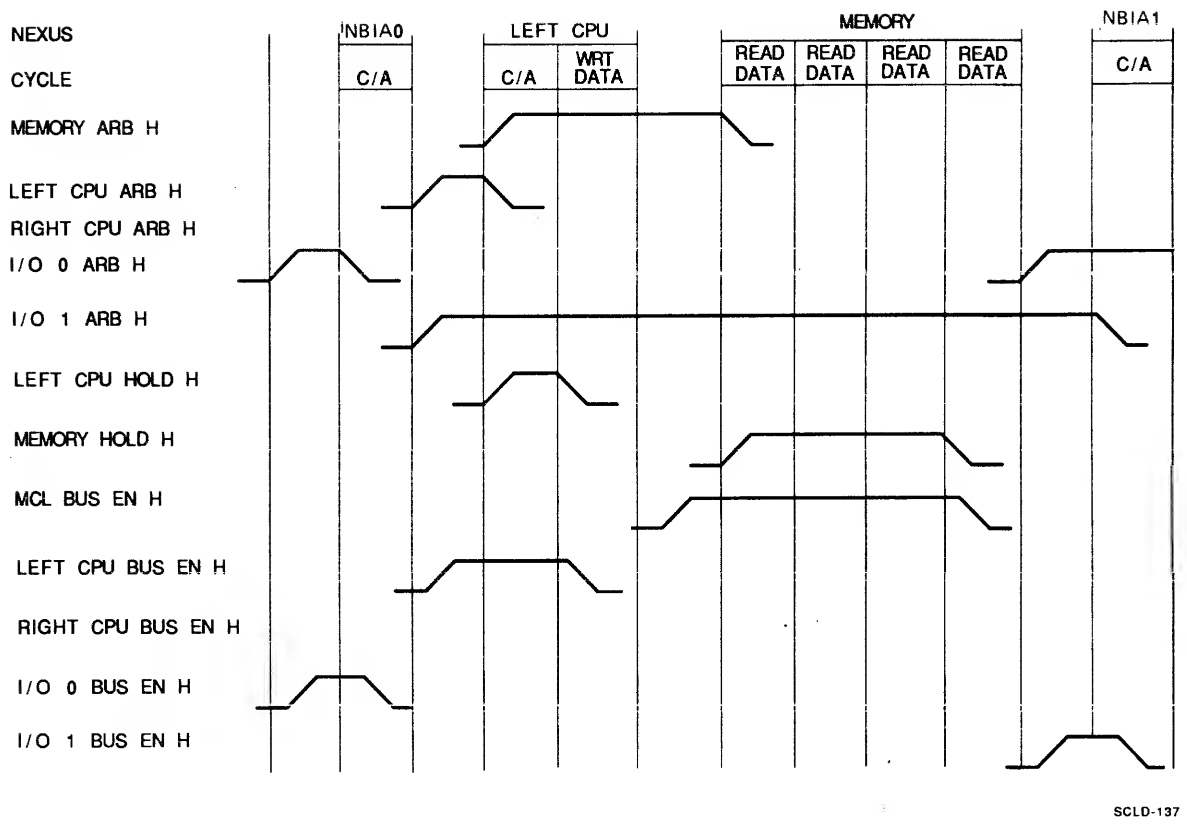


Figure 2-7 NMI Arbitration Line Timing (Typical)

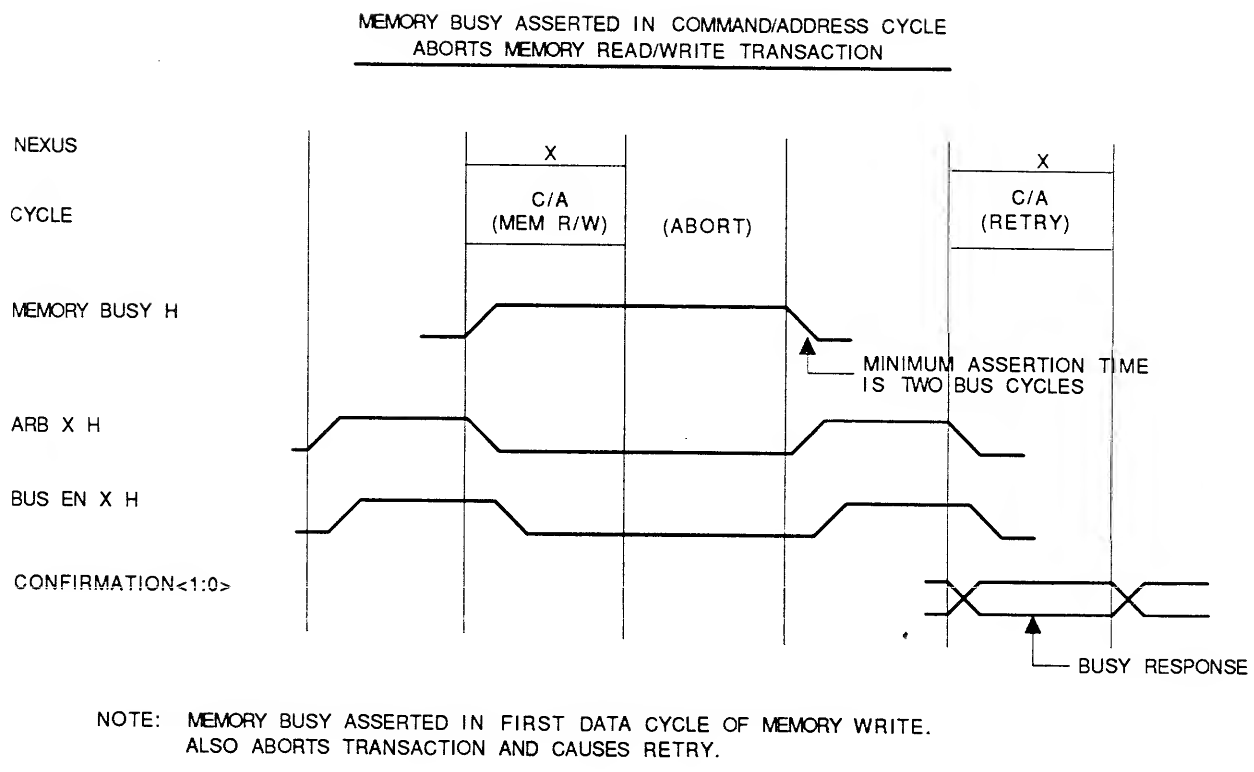

MEMORY BUSY ASSERTED IN COMMAND/ADDRESS CYCLE
ABORTS MEMORY READ/WRITE TRANSACTION
NEXUS
CYCLE
X
C/A
(MEM R/W)
(ABORT)
X
C/A
(RETRY)
MEMORY BUSY H
MINIMUM ASSERTION TIME
IS TWO BUS CYCLES
ARB X H
BUS EN X H
CONFIRMATION<1:0>
BUSY RESPONSE
NOTE: MEMORY BUSY ASSERTED IN FIRST DATA CYCLE OF MEMORY WRITE.
ALSO ABORTS TRANSACTION AND CAUSES RETRY.
SCLD-138

Figure 2-8 MEMORY BUSY Timing

2.1.6 NMI Interrupts

NMI interrupt lines connect to both CPUs in dual-processor systems. However, only the primary CPU is enabled to accept NMI interrupts. A CPU is enabled by setting bit <00> in its NMI interrupt enable register.

There are two types of CPU interrupt requests generated on the NMI.

1. Conventional interrupt requests by an I/O device (memory or an NBIA) when it has errors or other conditions to report to the CPU. There are four interrupt request levels, BR7 through BR4, where BR7 has the highest priority and BR4 the lowest.

2. A special CPU interrupt request caused by the assertion of the NMI FAULT line. FAULT not only interrupts the primary CPU, it freezes an NMI transaction silo in the CBox that holds the state of selected bus signals for the faulting and preceding bus cycles. All NMI nexus, including the CPU(s), can assert FAULT when certain types of bus errors are detected.

The interrupt priority level (IPL) assigned to each interrupt request asserted on the NMI is listed below.

Request (Device)	IPL
NMI Fault	1C
BR7 (NBIA0)	17
BR7 (NBIA1)	17
BR6 (NBIA0)	16
BR6 (NBIA1)	16
BR5 (NBIA0)	15
BR5 (NBIA1)	15
BR5 (Memory)	15
BR4 (NBIA0)	14
BR4 (NBIA1)	14

An NBIA makes conventional device interrupt requests by asserting an interrupt request signal and two other signals that specify the interrupt request level, BR7 through BR4. (The memory just asserts a request line; its request is always serviced by the CPU at a BR5 level.) The interrupt request by an NBIA can be generated internally (an NBI interrupt) or it can be a request generated by a VAXBI device. For internal interrupts, the interrupt level (as specified by the two NMI interrupt level signals) is BR4. For VAXBI device interrupts, the level (also asserted on the NMI) is the same as the VAXBI interrupt level. That is, the VAXBI also has four interrupt levels, BR7 through BR4.

There are two copies of the interrupt request signal asserted by an NBIA (and memory). The NBIA signals are called DEVN LINTR and DEVN RINTR. Also, there are two copies of the NBIA interrupt level signals, DEVN LINTR LVL <1:0> and DEVN RINTR LVL <1:0>. One copy connects to the left CPU in dual-processor systems and to the single CPU in single-processor (bounded) systems. The other copy connects to the right CPU in dual-processor systems.

When ready to service the highest priority interrupt request, and if the request is by an NBIA, the CPU microcode (by means of an NMI read transaction) reads one of the NBIA's four vector registers. There is a vector register for each of the four interrupt request levels, and its value is used by the CPU microcode to generate an interrupt vector address in the SCB. The vectors for locally generated interrupts by each NBIA are in page 0 of the SCB. In this case, an NBIA is interrupting as an NMI nexus like memory, whose vector is also in page 0. Vectors for VAXBI devices connected to the NBIAs are allocated starting with page 1. Refer to Section 1.4.3.

When a CPU or NBIA nexus detects an NMI fault, it asserts and negates a FAULT DETECT line. (Refer to Figure 2-9.) There is a FAULT DETECT line for each CPU and NBIA, and all the signals are logically ORed in the memory nexus causing it to assert a single NMI FAULT line at the beginning of the next bus cycle. (When the memory nexus detects a fault, it just asserts the FAULT line.) In the primary CPU, the FAULT line then generates the CPU interrupt request (the vector is in page 0 of the SCB), and it freezes the NMI transaction silo. Also, in all nexus (including the NBIA), it latches the NMI fault status bits so they can be examined during the interrupt sequence. Conditions causing an NMI FAULT are discussed in Section 2.1.7.

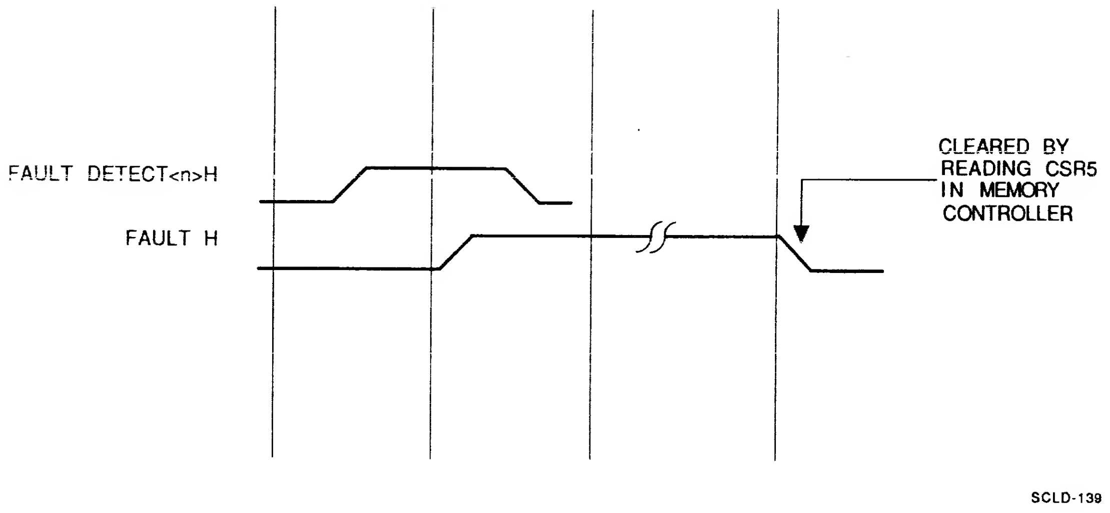


Figure 2-9 Fault Signal Timing

2.1.7 NMI Errors

There are two types on NMI errors: faults and those causing a conventional CPU interrupt request.

Faults are errors that (when detected by any nexus) cause the single NMI FAULT line to be asserted. This interrupts the CPU but it also freezes the NMI transaction silo. (For this class of errors, a recent history of NMI transactions is useful in determining the cause of error.) Errors causing a nexus to assert NMI FAULT are bus parity errors and read or write sequence errors. A write sequence error is when a responder does not receive enough write data to complete the transaction. Similarly, a read sequence error is when a commander does not receive enough return read data. Also, a read sequence error is caused when a READ CONTINUE code is received on the FUNCTION lines during a read cycle, and there was no RETURN READ DATA code received (flagging the first longword of read data) during a previous data cycle in the transfer.

NMI errors causing a conventional interrupt (by the nexus detecting the error) are all timeout errors. When a timeout is detected by the NBIA (and memory), the interrupt request is made using the NMI interrupt lines. When a timeout is detected by a CPU, the interrupt is internally generated.

Timeout errors occur in a CPU or NBIA when it cannot gain access to the NMI, or (if it wins the bus and while retrying a transaction) it continues to receive no response, or a BUSY or INTERLOCKED response, from the responder. Also, a timeout error is set in the CPU or NBIA if a read transaction is acknowledged by the responder but the responder does not return any read data. A timeout error occurs in the memory when it is interlocked by a read interlocked transaction but not unlocked by a write unlock transaction.

NMI error checking is summarized below.

NMI Error	Interrupt/ Fault	Nexus Checking Error CPU	I/O	Memory
No Return Read Data (Timeout)	I	YES	YES	NO
No Access To Bus (Timeout)	I	YES	YES	NO
No Access, Busy (Timeout)	I	YES	YES	NO
No Access, Interlocked (Timeout)	I	YES	YES	NO
No Access, No Response (Timeout)	I	YES	YES	NO
Interlock, No Unlock (Timeout)	I	NO	NO	YES
Bus Parity Error	F	YES	YES	YES
Write Sequence Error	F	NO	YES	YES
Read Sequence Error	F	YES	YES	NO

2.2 DATA BUS (BETWEEN NBIA AND NBIB)

The data bus interconnects the NBIA and an NBIB. The NBIA has two data bus ports (0 and 1) each connecting to one of the two NBIBs (0 and 1) that can be connected. Physically, a data bus consists of four cables that plug into the the NBIA and NBIB backplanes. Data bus signal levels are TTL.

Data bus signal descriptions are given in Table 2-2. Similar to the NMI, there are 32 multiplexed address/data lines. However, unlike the NMI, there are also multiplexed mask and function lines. Control lines include latch enables and read enables that allow command/address information and read/write data to be transferred to and from the data bus buffers in the NBIB under control of the NBIA. Also included are handshaking signals that start (and signal the end of) the CPU and DMA read/write operations that the NBIA and connected NBIBs perform.

Table 2-2 Data Bus Signals

Signal(s)	Number	Description
D <31:0> L	32	Multiplexed address/data lines.

Operation	Information Transferred	Transmitted By
CPU Write	VAXBI address/	NBIA
	write data	NBIA
CPU Read	VAXBI address/	NBIA
	read data	NBIB
DMA Write	Memory address/	NBIB
	write data	NBIB
DMA Read	Memory address/	NBIB
	read data	NBIA

Signal(s)	Number	Description
MF <4:0> L	5	Multiplexed mask(and status)/function lines.

Operation	Information Transferred	Transmitted By
CPU Write	NMI write function/	NBIA
	write mask	NBIA
CPU Read	NMI read function/	NBIA
	0s	NBIB
DMA Write	NMI write function/	NBIB
	write mask	NBIB
DMA Read	NMI read function/	NBIB
	read data status	NBIA

NOTE

Function transmitted by NBIA during CPU read is a modified IDENT command equal to 19 (hex) if reading interrupt vector.

Signal(s)	Number	Description
PD L	1	Transfers even parity bit for the D lines.
PMF L	1	Transfers even parity bit for the MF lines.

Table 2-2 Data Bus Signals (Cont)

Signal(s)	Number	Description
L <n> LTCH EN (n = 5,7,9,11)	4	Asserted by NBIA to latch command/-address information and read/write data into NBIB.
L <n> RD EN (n = 6,8,10,12)	4	Asserted by NBIA to read command/-address information and read/write data from NBIB.
INTR REQ <3:0>	4	Asserted by NBIB to indicate VAXBI interrupt request(s) received.
		INTR REQ <3:0> — VAXBI REQUEST LEVEL
		1XXX — BR7
		X1XX — BR6
		XX1X — BR5
		XXX1 — BR4
CPU REQ L	1	Asserted by NBIA during CPU read/write to initiate VAXBI read/write (or IDENT) transaction by NBIB.
CPU REQ PENDING	1	Asserted by NBIA before CPU REQ. Prevents NBIB from responding to a VAXBI read/write transaction (a DMA request) until VAXBI transaction initiated by CPU REQ is attempted.
NXT OR CPU DONE L	1	Asserted by NBIB during CPU read/write to indicate successful completion of VAXBI transaction.
CPU ERROR L	1	Asserted by NBIB during CPU read/write to indicate VAXBI transaction did not complete successfully.
BUF SEL L	1	Asserted by NBIB during DMA read/write to request that NBIA take command/-address (and write data) and initiate NMI read/write transaction.

Table 2-2 Data Bus Signals (Cont)

Signal(s)	Number	Description
REQ DONE L	1	Asserted by NBIA during DMA write to indicate it has taken command/address (and write data) from the NBIB. (The NBIA then initiates an NMI write transaction.) Asserted by NBIA during DMA read to indicate it has taken command/-address, initiated an NMI read transaction, and transferred one longword of return read data to the NBIB.
DMA ERROR L	1	Asserted by NBIA during DMA Read to indicate the NMI read transaction did not complete successfully.
NBIB PE L	1	Asserted by NBIB to indicate it detected a parity error for information (command/address or data) to be loaded in the BIIC for transmission on the VAXBI. (The BIIC regenerates parity when it transmits the information.)
FRC NBIB PE L	1	Asserted by NBIA (a CSR bit). Forces NBIB to assert NBIB PE.
FRC DMA BSY L	1	Asserted by NBIA (a CSR bit). Forces stall timeout error during DMA read/write.
BIIC LPBCK L	1	Asserted by NBIA (a CSR bit) to force a BIIC loopback request during a CPU read/write.

Table 2-2 Data Bus Signals (Cont)

Signal(s)	Number	Description
BI RESET L	1	Asserted by NBIB when it receives RESET from a connected VAXBI device. Causes NBIA to assert NMI RESET.
ADAPT INIT H	1	Asserted by NBIA to initialize NBIB and connected VAXBI devices. (NBIB generates ACLO/DCLO sequence on VAXBI.) Asserted by UNJAM console command or programmed NBI INIT (a CSR bit).
BI PWR UP L	1	Asserted by NBIB to indicate NBIB has just powered up or generated an ACLO/DCLO sequence in response to ADAPT INIT or RESET by a VAXBI device. (Sets CSR bit in NBIA.)
BUF NMI ACLO H	1	Asserted by NBIA when NMI ACLO is asserted. Causes NBIB to assert ACLO on the VAXBI.
BUF NMI DCLO H	1	Asserted by NBIA when NMI DCLO is asserted. Causes NBIB to assert BI DCLO on the VAXBI.
NBIB PRESENT H	1	Asserted by NBIB (when BI DCLO on the VAXBI is not asserted) to indicate it is inserted in the backplane. (Sets CSR bit in NBIA.)

2.3 VAXBI

The VAXBI is the system's I/O bus. Up to four VAXBIs can be connected depending upon the system configuration. (Each VAXBI connects to an NBIB, two NBIBs can be connected to an NBIA, and up to two NBIAs can be connected to the NMI in some systems.)

Each VAXBI is a 32 bit wide synchronous bus with parity that can interconnect up to 16 VAXBI interfaces (nodes). The processor node for each VAXBI in a system is the NBIB, which (together with the NBIA) interfaces the other nodes to the system's CPU(s) and memory that are connected to the NMI. Most VAXBI protocol (including bus arbitration) is performed automatically by a VAXBI node's BIIC, the ZMOS integrated circuit used by all nodes as the bus interface.

As for the NMI, data is transferred over the VAXBI by means of read/write transactions. However, there are additional transaction types (to generate interrupt requests and read interrupt vectors, for example). Also, there is distributed bus arbitration. Each node requesting use of the bus samples all bus requests, and the node with the highest priority then assumes control of the bus during the next bus cycle (when the bus is inactive), or following the current bus transaction (when the bus is busy). To allow bus arbitration when the bus is busy, an embedded arbitration cycle is included as part of every bus transaction.

A detailed description of the VAXBI is given in Chapter 2 of the System Bus Summary. A basic description follows.

2.3.1 VAXBI Signals

VAXBI signals are defined in Table 2-3. All signal levels are TTL except for the FET driven AC LO and DC LO signals and the ECL clocks.

Table 2-3 VAXBI Signals

Signal(s)	Number	Description
D<31:00> L	32	Multiplexed address/data lines. Specify length of transfer and 30-bit address during command/-address cycles of read, write, and invalidate transactions. (Also specify interrupt level and/or destination mask information during command/address cycles of other transactions.) Transfer write, read, or vector data during data cycles. Specify decoded IDs of arbitrating nodes during arbitration cycles.
I<3:0> L	4	Information lines. Specify VAXBI command during command/address cycles, byte mask during write data cycles, and data status during read and vector data cycles. Also specify the master's ID during embedded arbitration cycles.

I<3:0> (Hex)	Command (See note)
0	Reserved
1	Read (READ)
2	Interlocked Read with Cache Intent (IRCI)
3	Read with Cache Intent (RCI)
4	Write (WRITE)
5	Write with Cache Intent (WCI)
6	Unlock Write Masked with Cache Intent (UWMCI)
7	Write Masked with Cache Intent (WMCI)
8	Interrupt (INTR)
9	Identify (IDENT)
A	Reserved
B	Reserved
C	Stop (STOP)
D	Invalidate (INVAL)
E	Broadcast (BDCST)
F	Interprocessor Interrupt (IPINTR)

Table 2-3 VAXBI Signals (Cont)

Signal(s)	Number	Description
		I<3:0> Byte Mask XXX1 Write Byte 0 XX1X Write Byte 1 X1XX Write Byte 2 1XXX Write Byte 3 NOTE Broadcast transaction not currently used. I<3:0> Data Status 0X00 Reserved 0X01 Read Data 0X10 Corrected Read Data 0X11 Read Data Substitute 0X00 Reserved 1X01 Read Data, Don't Cache 1X10 Corrected read Data, Don't Cache 1X11 Read Data Substitute, Don't cache
P0 L	1	Parity line. Transfers odd parity bit for the I lines during embedded arbitration cycles and for the D and I lines during command/address cycles, decoded ID (IDENT) cycles, and during data cycles when read-/write (or vector) data is being transmitted.

Table 2-3 VAXBI Signals (Cont)

Signal(s)	Number	Description
CNF<2:0> L	3	Confirmation lines. Specify command/data response by slave(s) during the data cycles of a transaction. Also specify data response by node receiving data during the two bus cycles following the data cycles of a transaction. CNF <2:0> — Response 000 — No Acknowledgment (NOACK) 001 — Illegal 010 — Illegal 011 — Acknowledgment (ACK) 100 — Illegal 101 — STALL 110 — RETRY 111 — Illegal
NO ARB L	1	Asserted by arbitrating nodes, master, pending master, or slave to inhibit arbitration by nodes during the next bus cycle. The pending master is the node winning the bus after an embedded arbitration cycle.
BSY L	1	Asserted by master or slave to indicate a transaction is in progress. Can also be asserted by any node to extend the current transaction, or (when asserted together with NO ARB), delay the start of the next transaction in order to perform special mode operations such as a loopback request.
RESET L	1	Asserted by NBIB when it generates a VAXBI ACLO/DCLO sequence during NBIB initialization. Asserted by connected VAXBI device to bootstrap the system. (Causes NBIA to assert NMI RESET.)
BAD L	1	Indicates one or more nodes detected a selftest or other error. Not asserted or monitored by NBIB.
STF L	1	Enables fast selftest mode in all nodes. Not asserted or monitored by NBIB.

Table 2-3 VAXBI Signals (Cont)

Signal(s)	Number	Description
TIME H/L	2	20 MHz differentially driven (ECL) clock generated by NBIB. Used in conjunction with PHASE H/L to provide reference for VAXBI cycle timing in all nodes.
PHASE H/L	2	5 MHz differentially driven (ECL) clock generated by NBIB. Used in conjunction with TIME H/L to provide reference for VAXBI cycle timing in all nodes.
AC LO L	1	Indicates ac power is below specified limits. Asserted by NBIB (when NMI ACLO is asserted) and by VAXBI expander cabinet during powerup. Also asserted when NBIB is initialized by NBIA. (NBIB generates ACLO/DCLO sequence to simulate power-up/-power-fail condition.)
DC LO L	1	Indicates dc power is below specified limits. Asserted by NBIB (when NMI DCLO is asserted) and by VAXBI expander cabinet during powerup. Also asserted when NBIB is initialized by NBIA. (NBIB generates ACLO/DCLO sequence to simulate power-up/-power-fail condition.)

2.3.2 Basic Timing

The clocks and basic bus timing are shown in Figure 2-10. Signals on the data path and synchronous control lines are asserted and deasserted at the beginning of a bus cycle. The signals are received and latched near the end of the cycle. (The data path signal lines are the D, I, and P0 lines; the synchronous control lines are the CNF, NO ARB, and BSY lines.) The other control lines (AC LO, DC LO, RESET, STF, and BAD) are asserted and deasserted asynchronously with respect to the bus cycle.

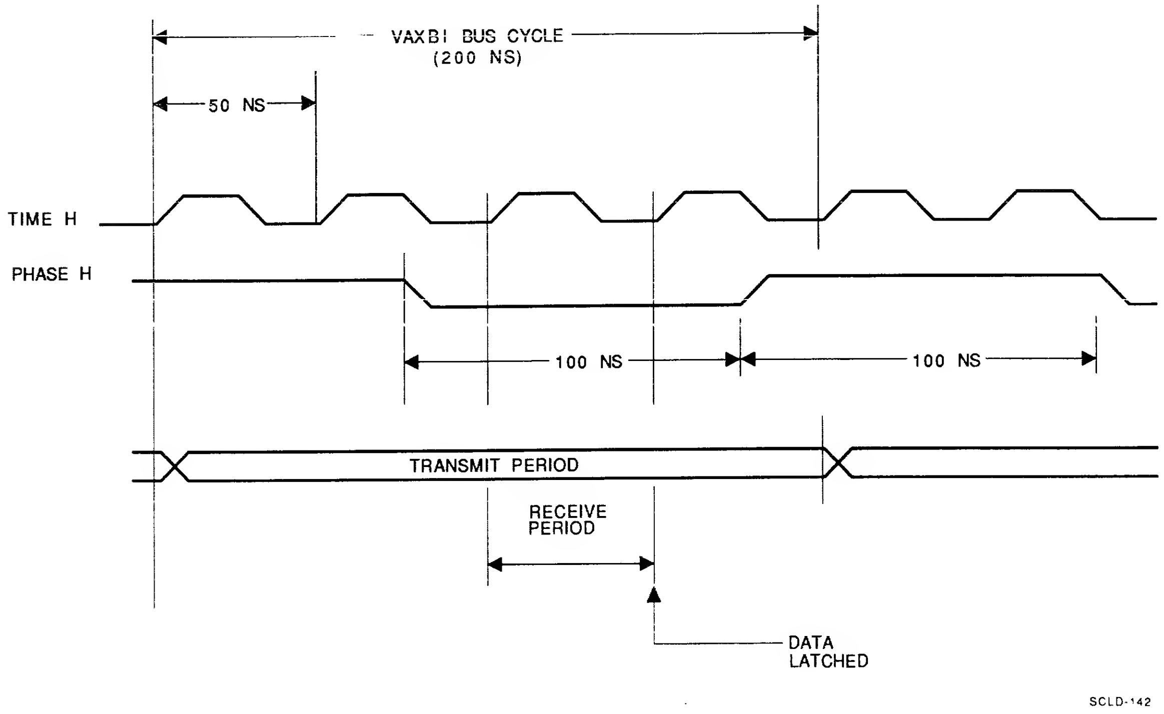


Figure 2-10 Basic VAXBI Timing

2.3.3 VAXBI Address Space

VAXBI address space, like NMI address space, is 1 Gbyte where one half is physical memory space and the other half is I/O space. Refer to Figure 2-11.

Memory space for a VAXBI in the system (except for that allocated to an I/O processor node's memory) is actually NMI address space. A VAXBI read/write transaction with a memory address, when responded to by the NBIB, causes an NMI transaction by the NBIA that reads or writes that address in NMI memory. The VAXBI address, when transferred from the NBIB to the NBIA and transmitted on the NMI, is not modified.

The allocated I/O space for a VAXBI is only 32 Mbytes. (The rest of the I/O address space is reserved.) The range of VAXBI I/O addresses, 2000 0000 to 21FF FFFF (Hex), is the same for each of the (up to four) VAXBIs in a system. This means that when VAXBI I/O space is accessed by an NMI transaction, address bits <26:25> must be cleared during the NMI to VAXBI address translation. The address bits are cleared by the NBIA before the address is sent to the NBIB.

NMI I/O Addresses (Hex)	VAXBI	VAXBI I/O Addresses (Hex) [Bits <26:25> Cleared by NBIA]
2000 0000 - 21FF FFFF	VAXBI0 (NBIB 0)	2000 0000 - 21FF FFFF
2200 0000 - 23FF FFFF	VAXBI1 (NBIB 0)	2000 0000 - 21FF FFFF
2400 0000 - 25FF FFFF	VAXBI0 (NBIB 1)	2000 0000 - 21FF FFFF
2600 0000 - 27FF FFFF	VAXBI1 (NBIB 1)	2000 0000 - 21FF FFFF

The 32 Mbytes of I/O space for each VAXBI consists of register space for each node, multicast space, node private space, and adapter window space. (Refer again to Figure 2-11.) Multicast space contains addresses for which more than one node can respond. Node private space contains registers that are not accessed from the VAXBI. For example, the NMI nexus registers in the NBIA (the CSRs and vector offset registers) have node private space addresses. Window space is used for address mapping by adapter nodes interfacing another bus and its devices to the VAXBI. That is, the VAXBI addresses are converted to addresses specific to the other bus (UNIBUS addresses, for example). A block of window space is allocated to each node.

The register space for each node (8K bytes) is shown in Figure 2-11. The first group of registers, called the VAXBI required registers, is implemented by every node on a VAXBI. The second group is the BIIC-specific registers. A register in this group may or may not be used depending upon the node design. All of the VAXBI-required and BIIC-specific registers (Figures 2-13 and 2-14) are contained in the node's BIIC. (BIIC register bit formats and descriptions are given in Chapter 1.)

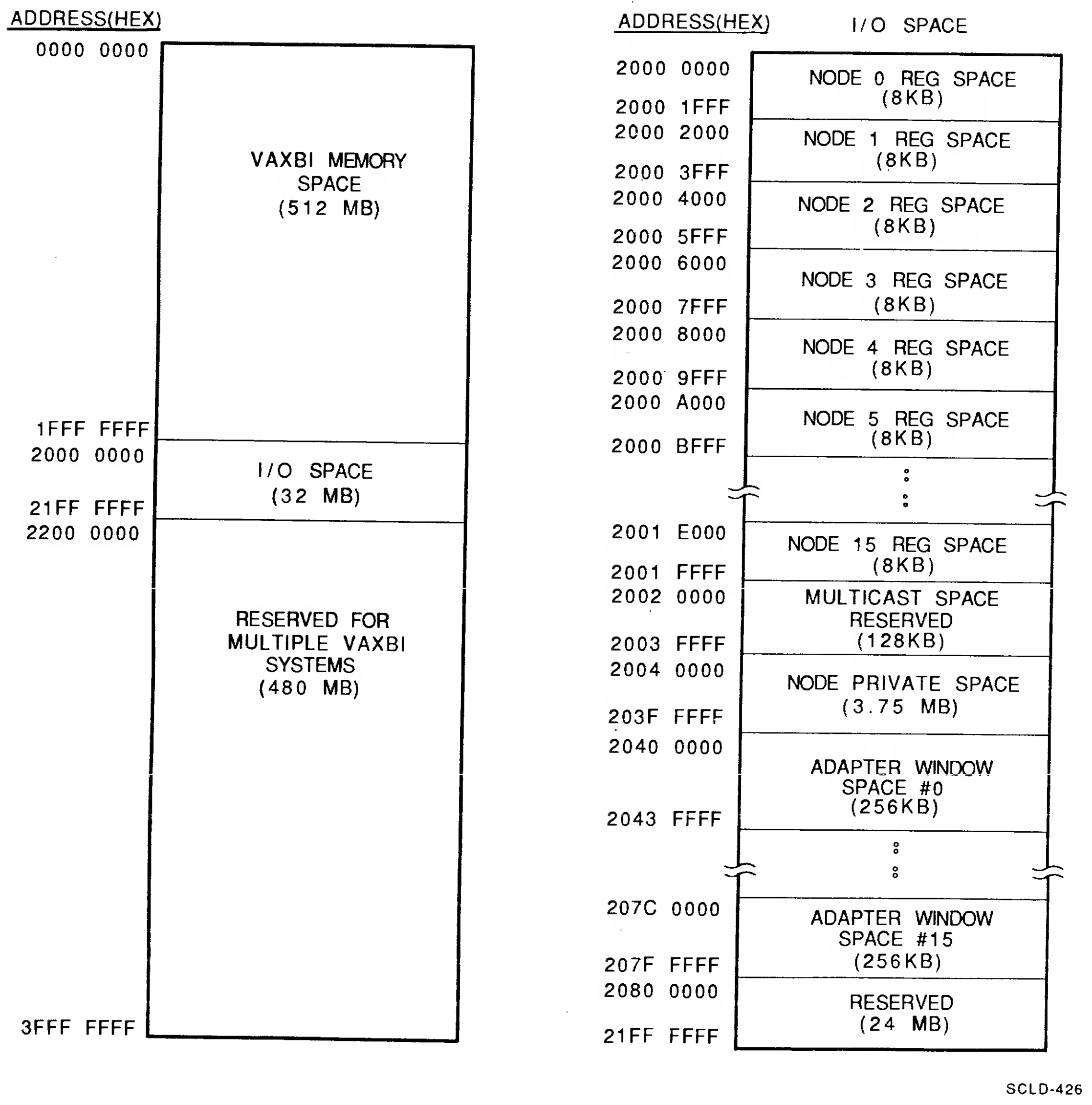


Figure 2-11 VAXBI Address Space

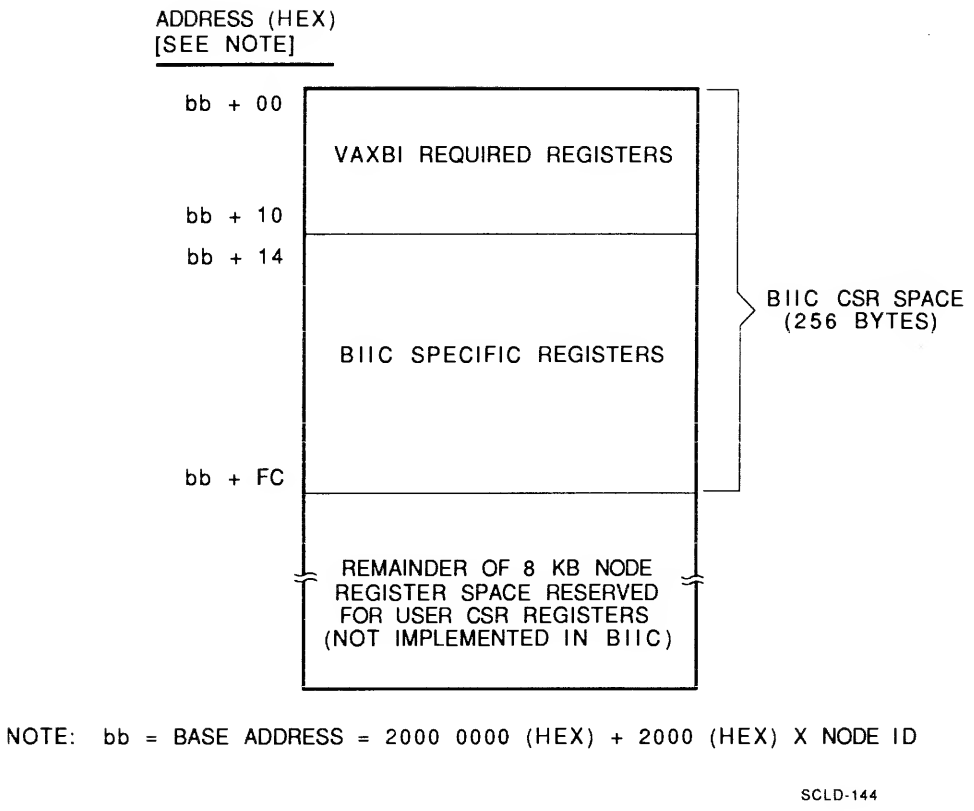

ADDRESS (HEX)
[SEE NOTE]
bb + 00
bb + 10
bb + 14
bb + FC
VAXBI REQUIRED REGISTERS
BIIC SPECIFIC REGISTERS
REMAINDER OF 8 KB NODE REGISTER SPACE RESERVED FOR USER CSR REGISTERS (NOT IMPLEMENTED IN BIIC)
BIIC CSR SPACE (256 BYTES)
NOTE: bb = BASE ADDRESS = 2000 0000 (HEX) + 2000 (HEX) X NODE ID
SCLD-144

Figure 2-12 VAXBI Node Register Space

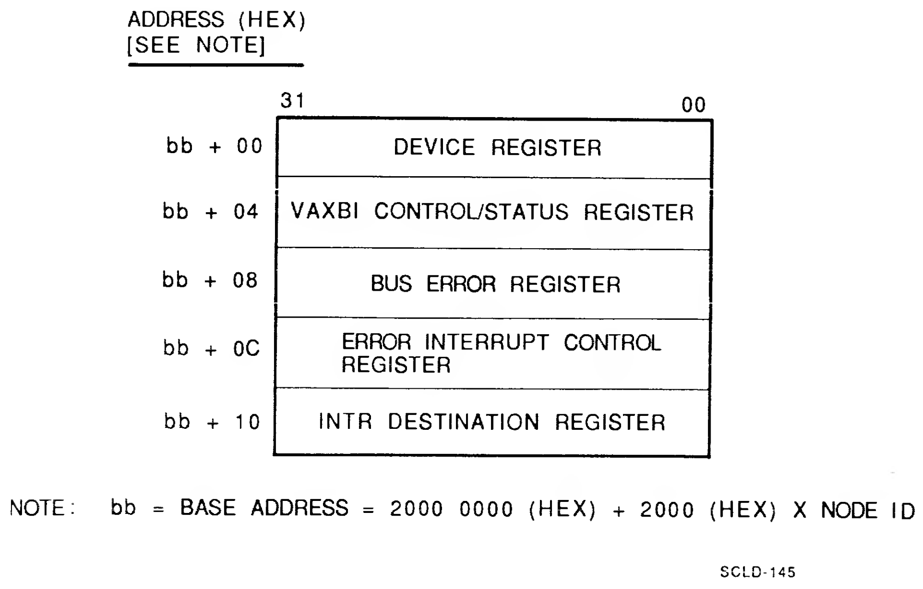

ADDRESS (HEX)
[SEE NOTE]
31
00
bb + 00
bb + 04
bb + 08
bb + 0C
bb + 10
DEVICE REGISTER
VAXBI CONTROL/STATUS REGISTER
BUS ERROR REGISTER
ERROR INTERRUPT CONTROL REGISTER
INTR DESTINATION REGISTER
NOTE: bb = BASE ADDRESS = 2000 0000 (HEX) + 2000 (HEX) X NODE ID
SCLD-145

Figure 2-13 VAXBI-Required Registers

ADDRESS (HEX) [SEE NOTE]	31 00
bb + 14	IPINTR MASK REGISTER
bb + 18	IPINTR/STOP DESTINATION REGISTER
bb + 1C	IPINTR SOURCE REGISTER
bb + 20	STARTING ADDRESS REGISTER
bb + 24	ENDING ADDRESS REGISTER
bb + 28	BCI CONTROL REGISTER
bb + 2C	WRITE STATUS REGISTER
bb + 30	FORCE IPINTR/STOP COMMAND REGISTER
bb + 34	UNUSED
bb + 38	UNUSED
bb + 3C	UNUSED
bb + 40	USER INTERRUPT CONTROL REGISTER
bb + 44 … bb + EC	UNUSED
bb + F0	GENERAL-PURPOSE REGISTER 0
bb + F4	GENERAL-PURPOSE REGISTER 1
bb + F8	GENERAL-PURPOSE REGISTER 2
bb + FC	GENERAL-PURPOSE REGISTER 3

NOTE: bb = BASE ADDRESS = 2000 0000 (HEX) + 2000 (HEX) X NODE ID

SCLD-146

Figure 2-14 BIIC-Specific Device Registers

2.3.4 VAXBI Read/Write Transactions

The VAXBI supports read/write longword, quadword, and octaword transactions. The NBIB responds to VAXBI read/write transactions addressing memory (DMA data transfers) provided the address falls within a range specified by the starting and ending address registers in the BIIC. Also, the NBIB originates read/write (longword) transactions when a VAXBI device register is addressed by a CPU read/write.

Read/write transaction timing is shown in Figures 2-15 and 2-16. Like all VAXBI transactions, the first cycle is a command/address cycle followed by an embedded arbitration cycle and at least one data cycle. For read/write transactions, there are one, two, or four data cycles depending upon the transaction length (longword, quadword, or octaword). Timing for read/write octaword transactions (four data cycles) is shown.

During the command/address cycle, the bus master (the node winning the bus and originating the transaction) transmits a 2-bit transaction length code and the 30-bit memory or I/O address on the D lines. It also transmits a read/write command on the I lines. The command, in addition to specifying a normal read/write, allows interlocked memory operations (interlocked reads/unlock writes). Read/write commands can also specify that data be cached. However, the NBIB interprets all commands as not having cache intent. Cached operations (by any I/O processors) are not supported on a VAXBI in this (NMI-based) system.

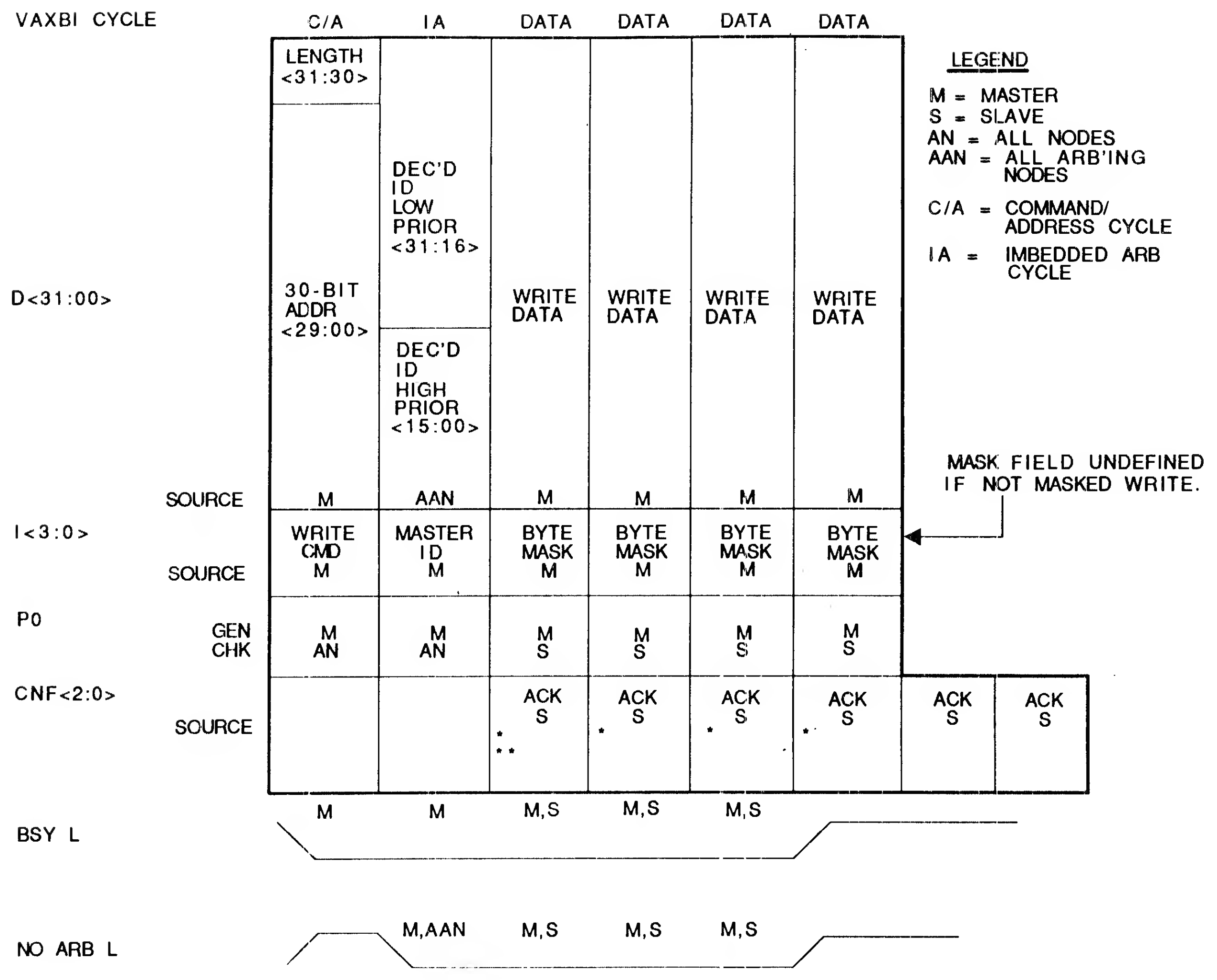
Following the embedded arbitration cycle (discussed in Section 2.3.8), and during each data cycle of a write transaction, the master transmits the write data on the D lines and the write mask (if any) on the I lines. Also, if it is ready and detects no errors, the addressed node (the slave) takes the write data and transmits an acknowledgment (an ACK code) back to the master on the CNF lines during each data cycle. An ACK response is also generated by the slave during the two bus cycles following the last data cycle.

After the embedded arbitration and during each data cycle of a read transaction, the selected slave (if it is ready and if it has detected no errors) transmits the read data back to the master on the D lines together with an ACK response on the CNF lines. It also transmits a read data status code on the I lines. The status code indicates if the data is valid, valid but corrected, or uncorrectable (READ DATA SUBSTITUTE code). (A READ DATA SUBSTITUTE code causes the master to set an error flag.) Also, to limit the caching and subsequent invalidation of data in some systems (but not in an NMI-based system like this one), the status code indicates when the read data is not to be cached. Like the write, an ACK response is transmitted on the CNF lines (this time by the master) for two cycles after the last data cycle.

If there is a NO ACK response (no responding slaves) during the first cycle of a read or write transaction, it probably indicates the master transmitted a nonexistent memory or I/O address. In this case, the master sets an error flag and ends the transaction.

If the selected slave is not ready to accept write data or return read data during any data cycle, it can assert a STALL code on the CNF lines. During stall data cycles for a write transaction, the master continues to transmit the same longword of write data on the D lines until the slave takes the data and generates an ACK response. For a read transaction, the D lines are undefined (may be in any state) until the stalled data and an ACK response are transmitted by the slave. If a slave has to generate 128 consecutive STALL responses during a read/write transaction, it sets an error flag, generates a NO ACK response, and ends the transaction.

A selected slave not ready to accept write data or send read data may not only stall. It may also cause the master to retry the transaction. A slave may cause a retry in the first data cycle or following a stall data cycle (if no data has previously been transferred) by transmitting a RETRY response on the CNF lines. If a slave causes a transaction to be retried 4096 times, the master sets an error flag.

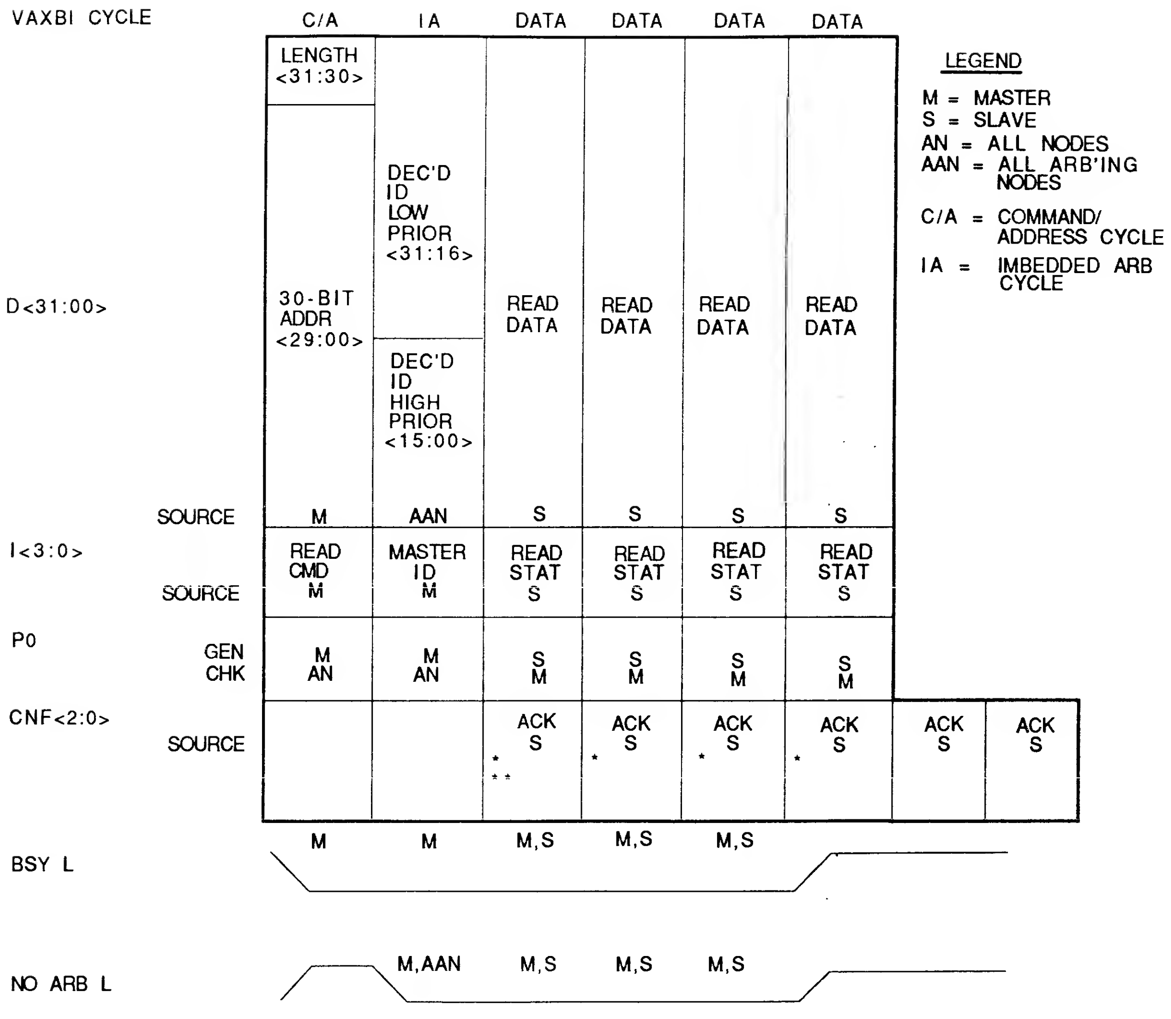


NOTES: 1. AN ASTERISK (*) INDICATES SLAVE MAY STALL (GIVE STALL RESPONSE) FOR ONE OR MORE BUSY CYCLES BEFORE TAKING DATA (ACK RESPONSE).

2. A DOUBLE ASTERISK (**) INDICATES SLAVE MAY REQUEST RETRY OF TRANSACTION (GIVE RETRY RESPONSE).

SCLD-185

Figure 2-15 VAXBI Write Transaction (Octaword Length)



NOTES: 1. AN ASTERISK (*) INDICATES SLAVE MAY STALL (GIVE STALL RESPONSE) FOR ONE OR MORE BUSY CYCLES BEFORE TAKING DATA (ACK RESPONSE).

2. A DOUBLE ASTERISK (**) INDICATES SLAVE MAY REQUEST RETRY OF TRANSACTION (GIVE RETRY RESPONSE).

SCLD-186

Figure 2-16 VAXBI Read Transaction (Octaword Length)

2.3.5 Interrupt Operation (INTR, IDENT, and IPINTR Transactions)

The interrupt (INTR) and identify (IDENT) transactions are used to signal and service conventional device type interrupts on the VAXBI. A node can send an interrupt request to one or more nodes using the INTR transaction. (As for the NMI, there are four request priority levels: BR4, 5, 6, and 7.) Then, when an interrupt fielding node is ready to service the interrupt requests at a specific request level, it uses an IDENT transaction to read an interrupt vector from the interrupting node. Because more than one node may be interrupting at that request level, the vector is read from the highest priority device based on the node ID.

The VAXBI also allows one processor node to interrupt another processor node. This is done using the interprocessor interrupt (IPINTR) transaction. Transaction format is similar to the INTR transaction. However, there is no need for a node responding to the interrupt request to follow with an IDENT transaction. This is because the interrupt fielding node stores the vector (and also the request level) information for this type of interrupt.

The interrupt fielding nodes on a VAXBI in this system are the NBIB and any I/O processors that are installed. The NBIB fields INTR transactions by passing the interrupt requests on to the NMI and the CPU. It then issues an IDENT transaction to collect the vector information when the CPU reads a vector register address in the NBIA. (There are four NBIA vector registers, one for each request level.) The interrupt requests generated on the VAXBI are normally from I/O devices but they may also be from an I/O processor node.

The NBIB can also field IPINTR transactions allowing an I/O processor to interrupt the CPU using an interprocessor interrupt request. (An I/O processor can also interrupt another I/O processor using an IPINTR.) Not only can the NBIB field an IPINTR, it can generate one. It can also generate an INTR allowing the primary CPU to interrupt an I/O processor with either type of interrupt request.

2.3.5.1 Interrupt (INTR) Transactions -- Format for the INTR transaction is shown in Figure 2-17. During the command/address cycle, the master (the interrupting node) transmits a 16-bit destination mask on the low-order D lines. Each bit in the mask corresponds to one of the 16 possible nodes on the VAXBI allowing the master to select one or more slaves to field the interrupt. For example, the master may signal both the NBIB and an I/O processor that it is interrupting.

The master also transmits the interrupt request level on four of the high-order data lines during the command/address cycle. There is a bit for each level as shown below. This allows a master (an adapter node, for example) to make an interrupt request at more than one request level with a single INTR transaction. (More than one I/O device attached to the adapter can be interrupting at the same time and at different assigned request levels.)

D<19:16>	Request Level
1XXX	BR7 (Highest Priority)
X1XX	BR6
XX1X	BR5
XXX1	BR4 (Lowest Priority)

During the single data cycle following the command/address and embedded arbitration cycles, any selected slave that intends to service the interrupt request asserts an ACK response on the CNF lines. This is essentially a command confirmation cycle with no data being transferred between the master and the selected slave(s). If no slave responds (a NO ACK response), the master sets an error flag.

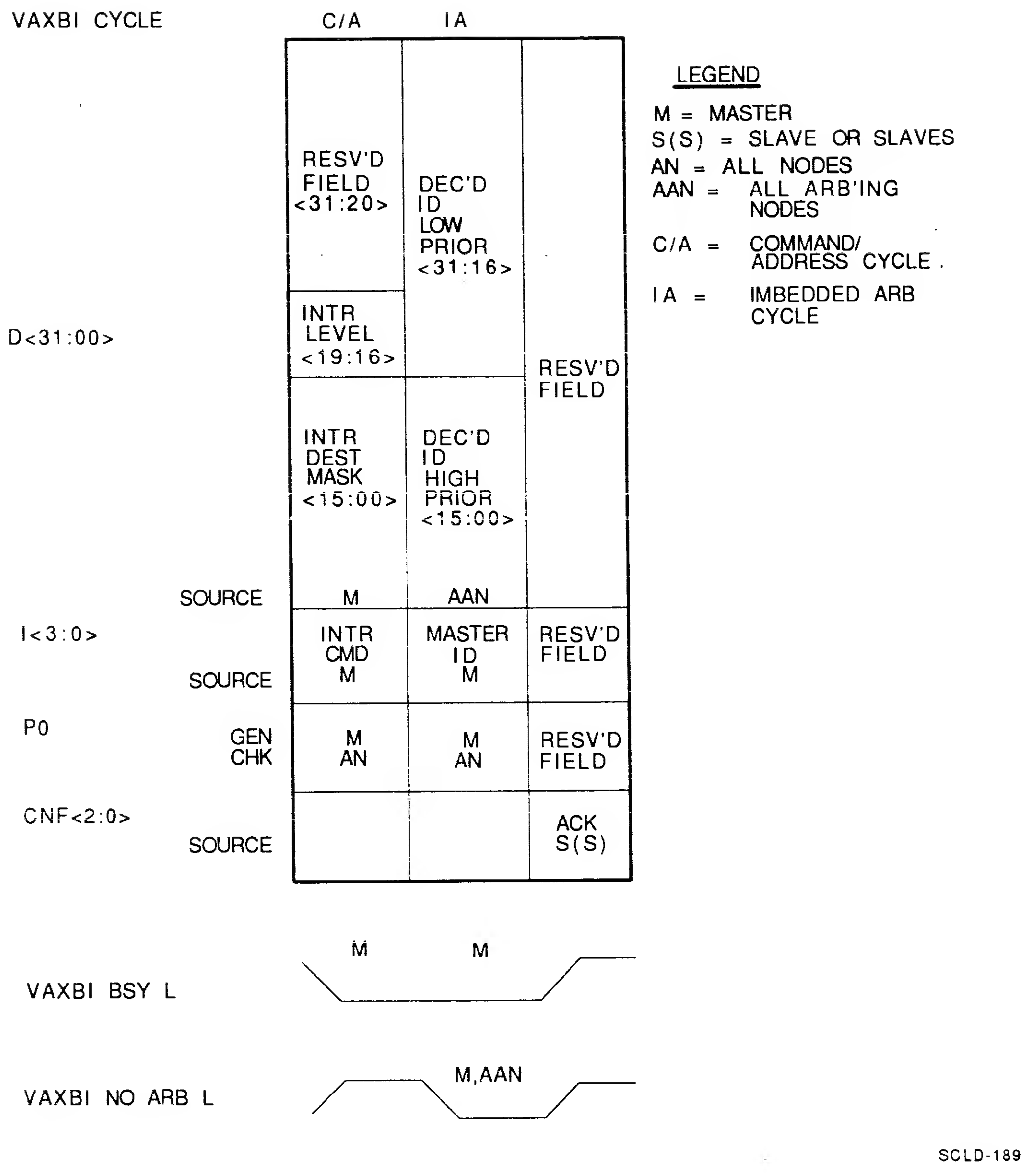


Figure 2-17 VAXBI Interrupt (INTR) Transaction

2.3.5.2 Identify (IDENT) Transactions -- As stated previously, an interrupt fielding node responding to an INTR transaction does an IDENT transaction to read an interrupt vector. IDENT format is shown in Figure 2-18.

During the command/address cycle and similar to the INTR transaction, the master transmits the interrupt level on four of the high-order D lines. However, this is the only select information transmitted and only one of the four lines should be asserted. The line asserted specifies the single interrupt request level to be serviced.

D<19:16>	Request Level
1000	BR7 (Highest Priority)
0100	BR6
0010	BR5
0001	BR4 (Lowest Priority)

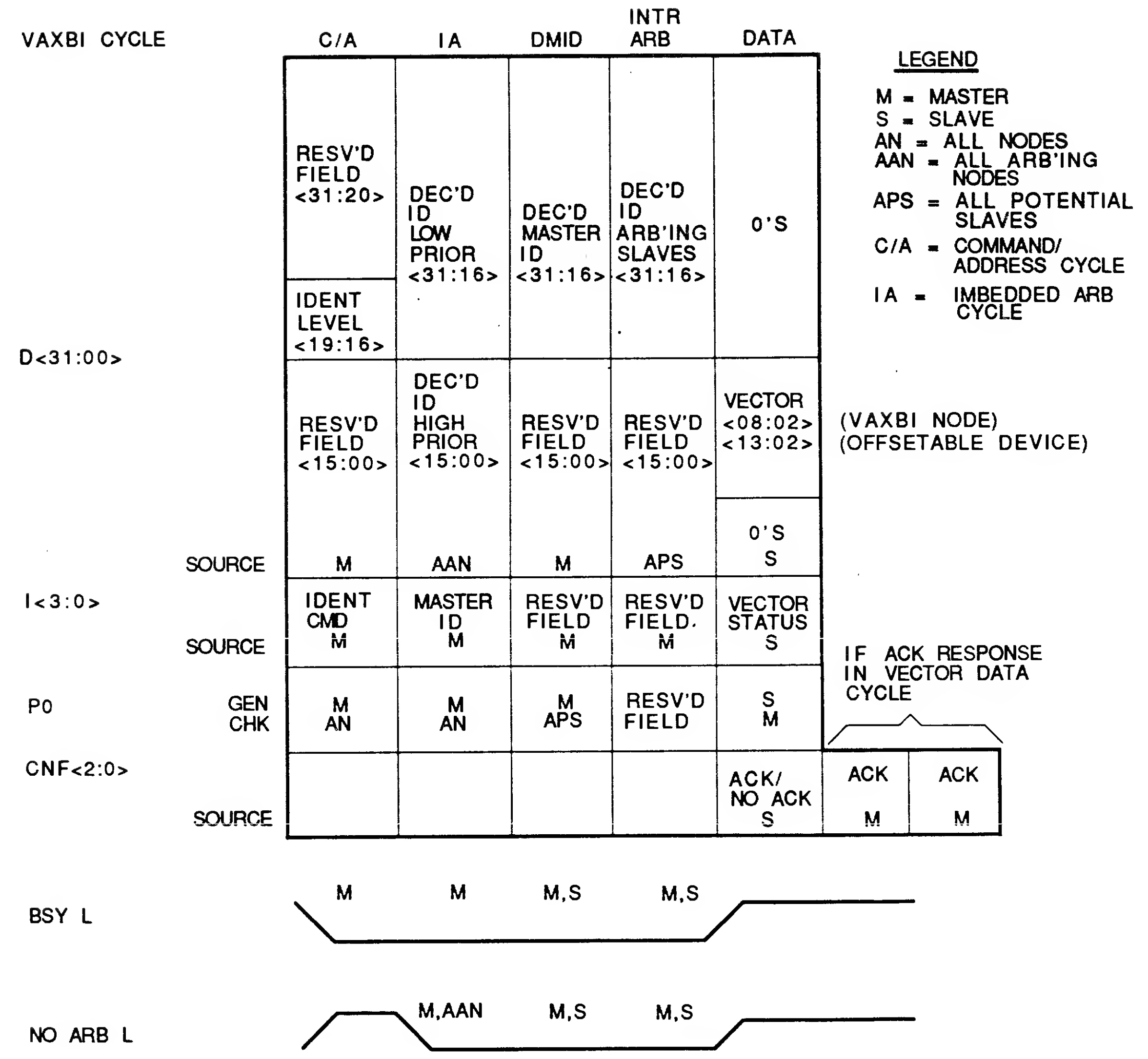
During the cycle following the command/address cycle and embedded arbitration cycle, the master transmits its decoded ID on the 16 high-order D lines. Only the D line corresponding to the master's node ID (plus 16) will be asserted. The transmission of the masters ID is necessary because there can be more than one interrupt fielding node on the bus, and nodes with an interrupt pending at the specified request level may not want service by the node doing the IDENT.

During the next bus cycle of the IDENT, the nodes wanting interrupt service by the master must arbitrate for the bus as when arbitrating to become bus master (Section 2.3.8). (Decoded IDs are transmitted only on the high-order data lines, however.) The nodes must arbitrate because only one can be serviced by the IDENT. That is, only one node can become the slave and return an interrupt vector to the master. Nodes not winning the arbitration must make another interrupt request (INTR transaction) unless they are serviced by another IDENT before a request can made.

After winning the bus, the slave (if it is ready) asserts an ACK response on the CNF lines and transmits the vector on the D lines during the next bus cycle. Also, similar to a read transaction, the slave may transmit a STALL response on the CNF lines for one or more cycles until the vector is ready to transmit. (An adapter node would do this if the vector had to be first read from an attached I/O device.) Again, 128 stall cycles will cause the slave to set an error flag and end the transaction. Once the vector is sent by the slave, the master transmits an ACK response on the CNF lines for the next two bus cycles if the transaction was executed correctly. If an error is detected, a NO ACK response causes an error flag to be set in the slave.

A node serviced by an IDENT may not necessarily have made a previous interrupt request. The interrupt condition may have occurred just before the IDENT was issued and before the node could arbitrate for the bus and initiate an INTR transaction. For this reason, a node cannot arbitrate for the bus to do an INTR transaction during the embedded arbitration cycle of the IDENT. It may win the bus and become pending bus master, and then it may win the following arbitration for service by the IDENT. It would then have to abort the INTR transaction because the request had already been serviced.

A NO ACK response occurring during (not after) an IDENT transaction is not an error condition. This can happen if the IDENT is the result of an interrupt request to more than one interrupt fielding node, and the request has already been serviced by another node.



NOTE: A NO ACK BY SLAVE IS A VALID RESPONSE INDICATING A TRANSIENT INTERRUPT OR THE INTERRUPT HAS BEEN SERVICED BY ANOTHER NODE. ALSO, SLAVE MAY STALL (GIVE STALL RESPONSE) FOR ONE OR MORE BUS CYCLES BEFORE RETURNING VECTOR (ACK RESPONSE).

SCLD-190

Figure 2-18 VAXBI Identify (IDENT) Transaction

2.3.5.3 Interprocessor Interrupt (IPINTR) Transactions -- Format for the IPINTR transaction (Figure 2-19) differs from the INTR transaction only in the selection information transmitted by the master in the command/address cycle. A destination mask is transmitted on the low-order D lines, but no interrupt level is transmitted on the high-order D lines. This is because the interrupt fielding node (the slave) holds the interrupt request level. Instead, the master's decoded ID is transmitted on the high-order D lines as in the third cycle of an IDENT. This is necessary because a slave may not be enabled to accept interprocessor interrupt requests from the current master.

As for the INTR, the single data cycle following the embedded arbitration cycle is essentially a command confirmation cycle. (No data is transferred between master and slave.) At least one slave must respond with an ACK during the cycle or an error flag is set in the master.

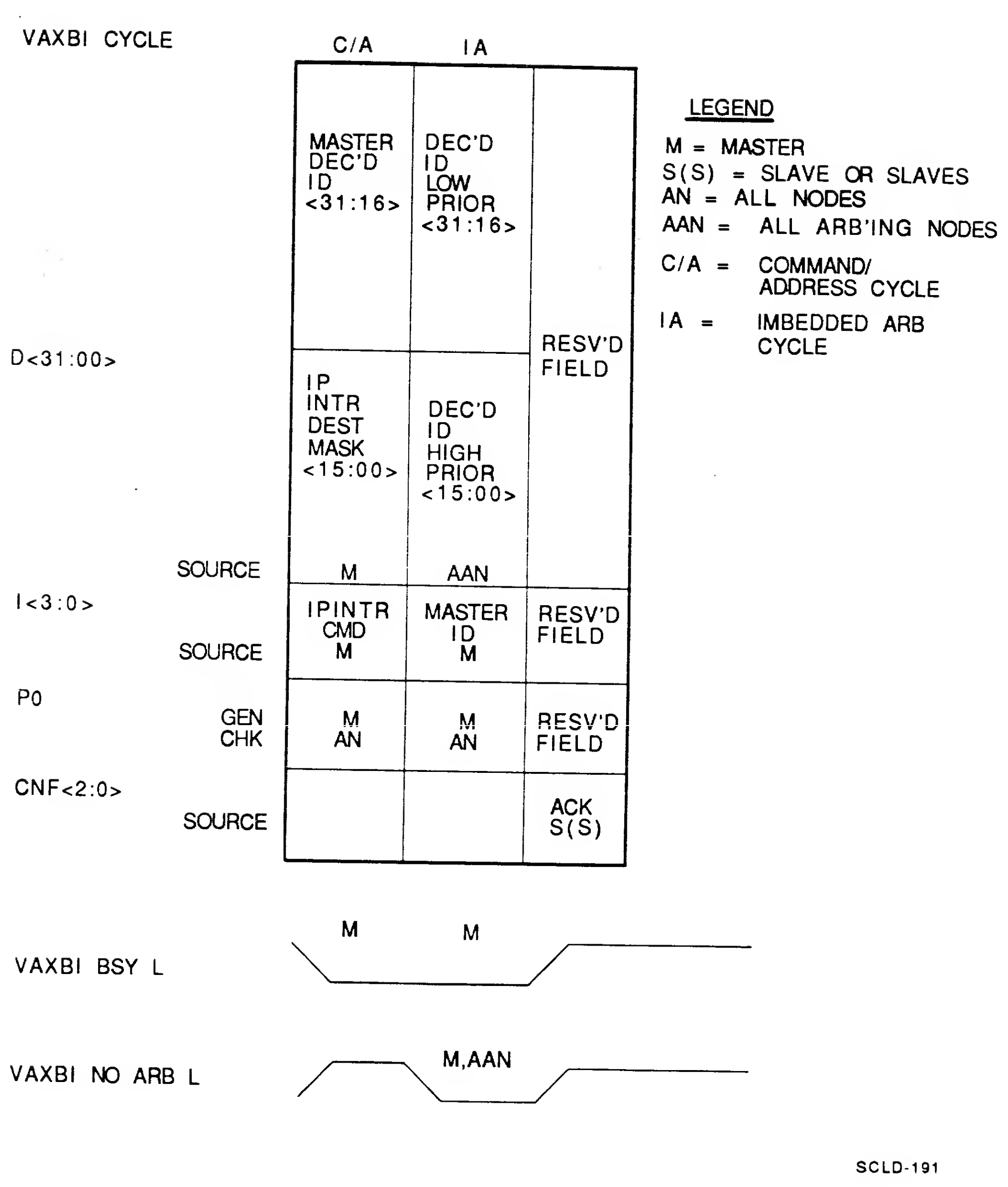


Figure 2-19 VAXBI Interprocessor Interrupt (IPINTR) Transaction

2.3.6 STOP Transactions

The STOP transaction is used by processor nodes (the NBIB, for example) to force the other nodes into a state where they cannot issue any more VAXBI transactions while retaining as much error and other status information as possible. However, nodes must still be able to respond to VAXBI transactions so that the retained status information can be examined by another node. A node can be returned to normal operation by forcing a selftest operation.

STOP transaction format (Figure 2-20) is similar to INTR and IPINTR transaction format except that the master transmits only a transaction destination mask on the D lines during the command/address cycle. More than one slave may be addressed and at least one must generate an ACK response on the CNF lines or the master sets an error flag. Like the INTR and IPINTR transactions, the response is generated in the single data cycle following the embedded arbitration cycle. (No data is actually transferred over the D lines in this cycle.)

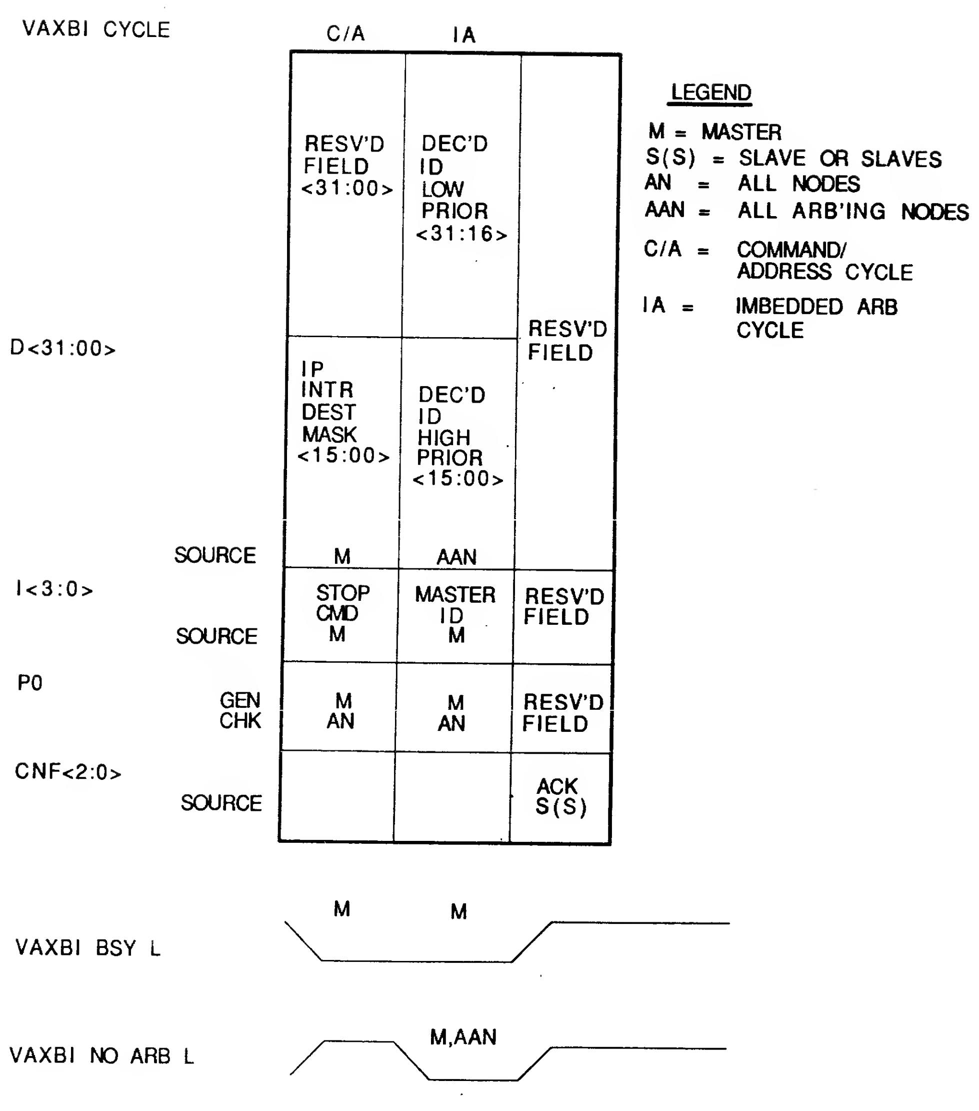


Figure 2-20 VAXBI STOP Transaction

2.3.7 Invalidate (INVAL) Transactions

The invalidate transaction (Figure 2-21) allows a processor node to signal other nodes that they may have cached data that is no longer valid. The bus master transmits the address and data length of the invalid block of data in the command/address cycle. More than one slave may respond with an ACK response on the CNF lines during the transaction's single data cycle. (No data is actually transferred and parity is not generated or checked during this cycle.) If there is no response by any node, an error flag is set in the master.

The invalidate transaction is not used on a VAXBI connected to this (NMI-based) system. Any processor nodes must have their caches turned off as stated previously. This is because the addresses of memory transactions local to the NMI are not passed to the VAXBI, and thus caches on the VAXBI (if turned on) could contain invalid data.

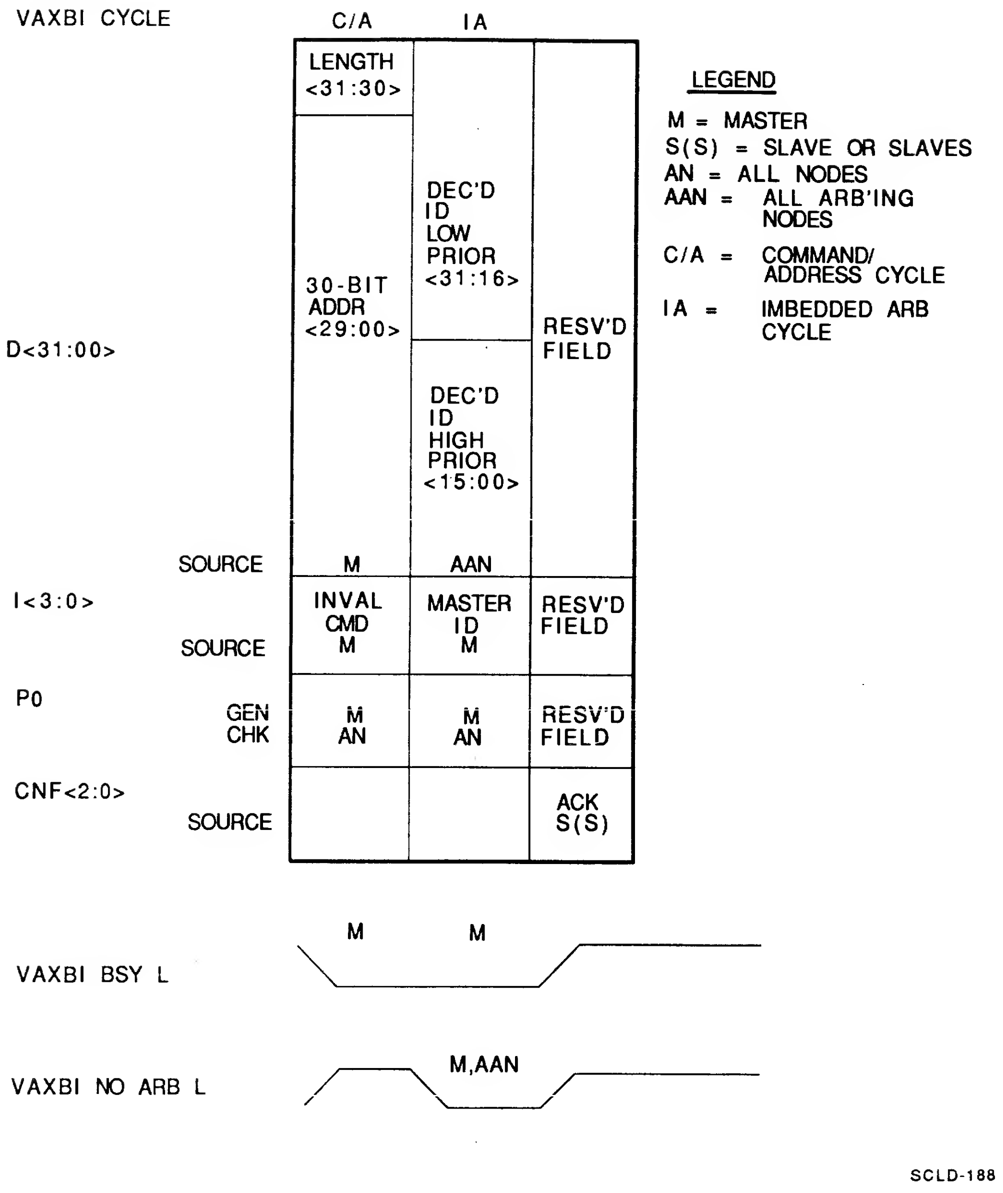


Figure 2-21 VAXBI Invalidate (INVAL) Transaction

2.3.8 Bus Arbitration

A node may request the bus during any bus cycle when the bus is inactive. It may also request the bus during the embedded arbitration cycle of any VAXBI transaction when the bus is busy. Each requesting node monitors all requests and, if it has the highest priority, assumes control of the bus as bus master either in the next cycle (when there is no transaction in progress) or at the end of the current transaction. If a node wins the bus during an embedded arbitration cycle, it is called the pending bus master until it assumes control of the bus.

2.3.8.1 Bus Requests -- A node requests use of the bus by asserting a D line that corresponds to its node ID. As shown in Figure 2-22, the line asserted may be one of the 16 high-order lines or one of the 16 low-order lines. The low-order lines are high priority requests. The high-order lines are low priority requests. Within each group of high or low priority requests, the node with the lowest ID (lowest numbered D line asserted) has the highest priority. Of course, any high priority request always has a higher priority than any low priority request.

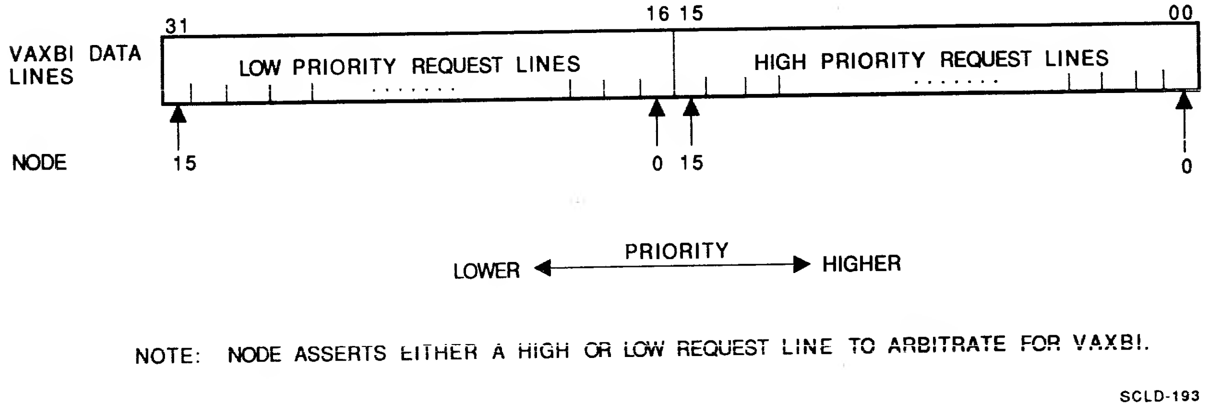

VAXBI DATA LINES
31
16 15
00
LOW PRIORITY REQUEST LINES
HIGH PRIORITY REQUEST LINES
NODE
15
0 15
0
PRIORITY
LOWER
HIGHER
NOTE: NODE ASSERTS EITHER A HIGH OR LOW REQUEST LINE TO ARBITRATE FOR VAXBI.
SCLD-193

Figure 2-22 Bus Arbitration Request Lines

2.3.8.2 Arbitration Modes -- Whether the node asserts its high priority line or its low priority line depends on the arbitration mode. There are three modes.

1. Dual round robin

2. Fixed high priority

3. Fixed low priority

Nodes are normally programmed to operate in dual round robin mode. In this mode, a requesting node asserts its high priority request only if the previous bus master has a lower ID (higher priority) than it does. At any other time, it asserts its low priority line. All nodes store the ID of the previous bus master, which is asserted on the I lines during the embedded arbitration cycle of the previous transaction.

On the average, dual round robin mode ensures equal access to the bus for all nodes. If a node requires rapid access to the bus as in some special real-time applications, it can be programmed to operate in fixed high priority mode. That is, it will always assert its high priority request line when requesting the bus. Also, access may be further enhanced by programming other nodes to operate in fixed low priority mode (only low priority requests will be asserted).

2.3.8.3 Arbitration Control -- The arbitration for the bus by any VAXBI node is illustrated in Figure 2-23. As shown, arbitration is controlled by two bus signals, NO ARB and BSY.

NO ARB is asserted by any nodes requesting the bus. It is also asserted by the current and pending bus masters and the slave at various times during a transaction depending upon the transaction type, its length, and the responses generated. The net result is that NO ARB is always asserted except during the following bus cycles.

1. Null cycles (no bus arbitration or transaction in progress)

2. Command/address cycles

3. The last data cycle in transactions unless there is a pending bus master

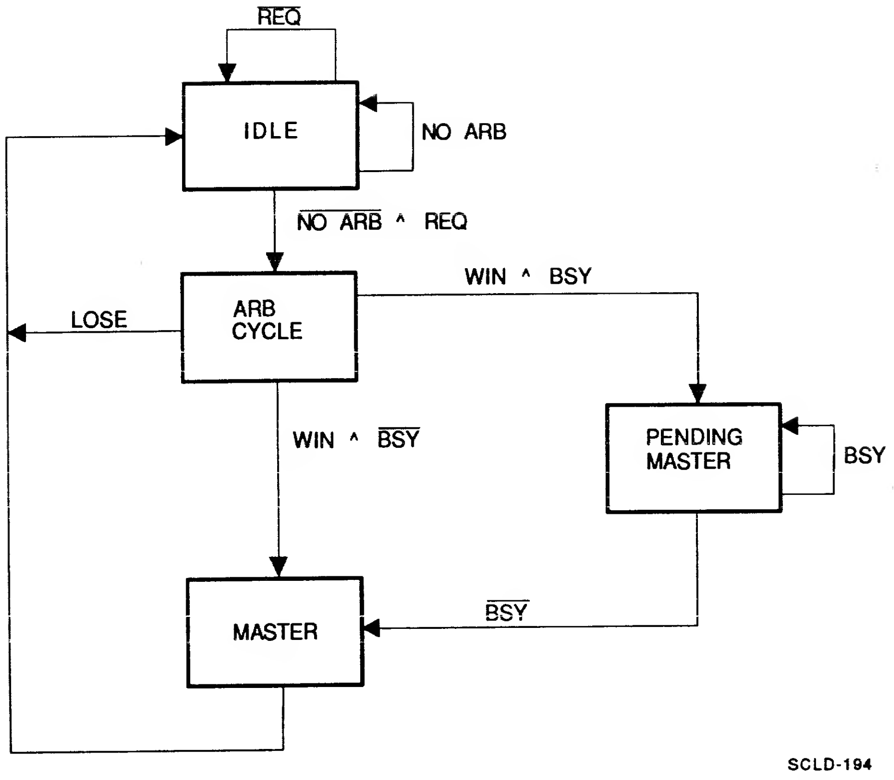


Figure 2-23 Arbitration State Diagram

BSY is asserted only by the current master and slave during a transaction. The master asserts BSY during the command/address and embedded arbitration cycles. Both the master and slave can assert BSY during data cycles as for the NO ACK signal. BSY can also be asserted to extend a transaction and, together with NO ARB, during special mode functions as well (Section 2.3.8.5). BSY is not asserted during the following bus cycles.

1. Null cycles

2. Arbitration cycles (when there is no transaction in progress)

3. The last cycle of transactions or special mode functions

When NO ARB is not asserted, nodes may arbitrate for the bus during the next cycle. When BSY is not asserted, a node may begin a transaction in the next cycle. For example, a node winning a bus arbitration when BSY = 0 becomes bus master and begins the transaction in the next cycle. (Refer again to Figure 2-22.) Similarly, if a node wins the bus during the embedded arbitration cycle when BSY = 1, it must wait as pending bus master until BSY = 0 before it can begin its transaction in the next cycle. Timing for NO ARB and BSY during typical bus operations is shown in Figure 2-24.

2.3.8.4 Extending a Transaction -- BSY may be asserted by a node to extend a transaction one or more cycles beyond its normal length. The additional cycles, called busy stall cycles, stop bus activity until the node is ready to respond properly to another transaction. For example, a node responding to the previously discussed STOP transaction may assert BSY to delay the start of the next transaction until it can enter STOP mode.

2.3.8.5 Special Mode Functions -- Both BSY and NO ARB may be asserted by a node during execution of some special mode functions. A bus transaction between other nodes may or may not be in progress. As when extending a transaction, the assertion of BSY delays the start of the next bus transaction until the special mode function is completed. (The NO ARB signal is asserted to identify the operation as a special mode function.) For example, a node doing a BIIC loopback operation cannot respond to bus transactions and it asserts BSY and NO ARB to prevent any transaction until the loopback is complete. A loopback operation is when the node is reading or writing one of its own BIIC registers using internal data paths only.

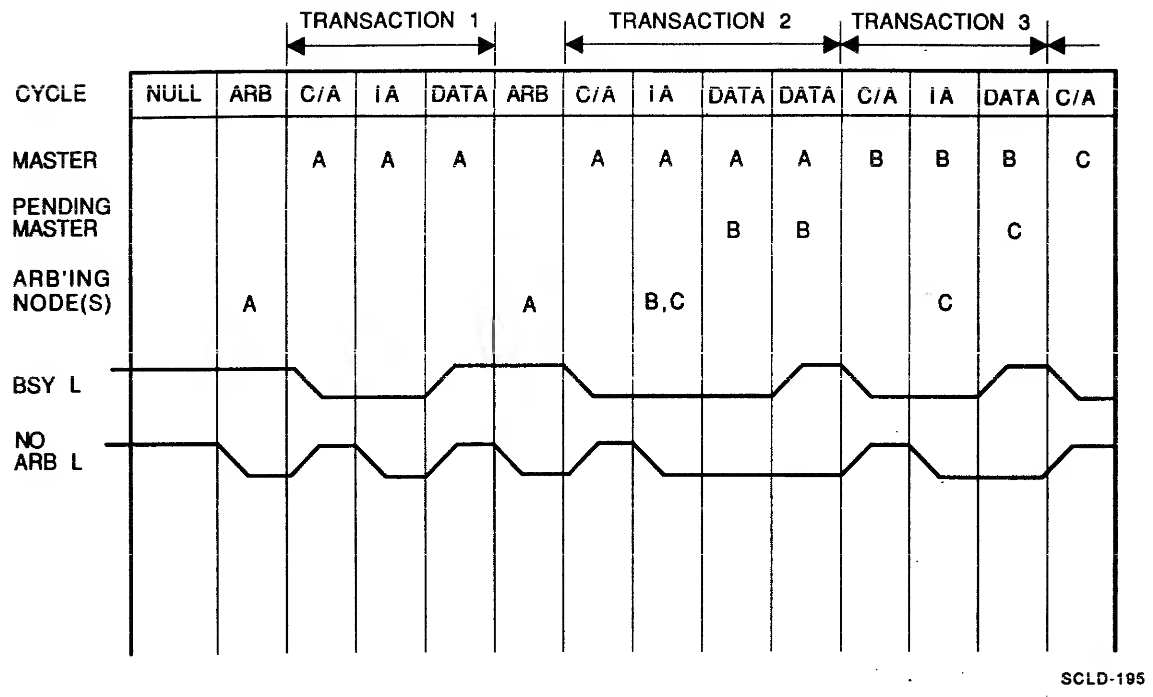


Figure 2-24 VAXBI Arbitration (Example)

2.3.9 VAXBI Errors
Required error detection by VAXBI nodes (done by the BIIC) consists of the following:

1. Parity Checking
2. Transmit Check Error Detection
3. Protocol Checking

2.3.9.1 Parity Checking -- The VAXBI has a single parity l
Except for embedded arbitration cycles, the parity bit (w
generated) is for the information on the D and I lines.
embedded arbitration cycles, only I-line parity is genera
nodes generating and checking parity during VAXBI cycles a
below.

Cycle	Parity Generating Node	Parity Checking Node(s)	Tr ac Ab
Command/Address	Master	All	Nc
Embedded Arbitration	Master	All	Nc
Decoded ID (IDENT)	Master	Potential Slave(s)	Nc
Write Data	Master	Slave	Ye
Read (Vector) Data	Slave	Master	Ye
Null	N/A	All (see note)	N/

NOTE: All nodes check that no D or L lines are asserted du
null cycle.

When a node detects a parity error, it sets an error flag
bus error register (in the BIIC) and generates an interrup
if error interrupts are enabled. Nodes detecting bad pari
command/address cycles do not acknowledge (A
command/address. Also, interrupting nodes that detect
error during the decoded ID cycle of an IDENT transacti
participate in the IDENT arbitration cycle.

2.3.9.2 Transmit Check Error Detection -- There are two
transmit check errors. One type is when the ir
transmitted by a master on the D, I, and P0 lines does not
with the information received (by the same node). The
made during command/address, write data, and decoded IL
cycles when the master is the only node transmitting infor
the D, I, and P0 lines. When the information transmi
received does not compare, the master sets an error flag i
error register and generates an interrupt request (if
The transaction is also aborted. The other type of trans
error is when a master or a slave should be asserting BS
ARB, and it does not detect the asserted state on the VAX
Again, the node sets an error flag in its bus error regi
generates an interrupt request (if enabled). Howeve
transaction is in progress, it is not aborted.

2.3.9.3 Protocol Checking -- The following errors in VAXBI transaction execution are checked. When an error is detected by a node, it sets an error flag in its bus error register and generates an interrupt request (if enabled).

1. NO ACK to Multi-Responder Command Received - A master received a NO ACK response for an INVAL, STOP, INTR, or IPINTR command.

2. Interlock Sequence Error - A node successfully completed an unlock write transaction that was not preceded by a corresponding read interlocked transaction.

3. IDENT Vector Error - An ACK response was not received by the master indicating the vector was not correctly received by the slave.

4. Read Data Substitute Error - A read data substitute (or reserved) status code was received with read (or vector) data and no parity error was detected.

5. Retry Timeout - The master received 4096 consecutive RETRY responses from the slave for the same transaction.

6. Stall Timeout - The slave transmitted 128 consecutive STALL responses (causes the slave to abort the transaction).

7. Bus Timeout - A node was unable to start a pending bus transaction after 4096 consecutive bus cycles.

8. Nonexistent Address - A master received no response to a read/write type command, and it detected no parity error or transmit check error for the command/address information (causes the master to abort the transaction).

9. Illegal Confirmation Error - A master or slave detected a reserved or illegal response code.

CHAPTER 3
FUNCTIONAL DESCRIPTION

3.1 INTRODUCTION

This chapter describes the major logic elements in the NBI's data path and control logic. It then gives a detailed functional description of how the major logic elements interact (at the input/output signal level) to perform the basic NBI operations. Circuit operation within each logic element (within the MCA, BIIC, etc.) is not discussed.

3.1.1 NBIA Block Diagram

The NBIA (Figure 3-1) consists of the following major logic elements:

- NMI data buffer
- NPAR MCA
- NBIM MCA
- NBFD MCA
- NBAP MCA
- NBCT MCA
- DSEQ MCA
- Translation buffer
- Data (bus) buffers
- Data bus (and transaction buffer) control sequencers

3.1.1.1 NMI Data Buffer -- The NMI data buffer consists of four MCAs, ND <3:0>. Each ND MCA is an 8-bit slice of the data path between the NMI and the EBus. (The EBus is the bus connecting the NMI data buffer to the DC022 transaction buffer.)

The data buffer contains two 32-bit latches for holding the address and data received from the NMI, and a 32-bit EBus transmit register for holding that information while it is transmitted (on the EBus) to the transaction buffer. The first latch, called the NMI receive latch, is latched by the trailing edge of A CLK. The second latch, which holds the address/data piped forward from the first, is latched by the trailing edge of B CLK. The EBus transmit register is loaded from the second latch by the trailing edge of A CLK.

Another register and a single latch (both 32 bits) are used for transferring address/data in the opposite direction, from the EBus to the NMI. An EBus receive register holds the address/data read from the transaction buffer, and the latch holds the address/data while it is transmitted on the NMI. The EBus receive register is clocked by A CLK (the leading edge). The latch, called the NMI transmit latch, is latched by the trailing edge of B CLK.

The NMI data buffer also contains the NBIA's CSRs, logic to decode the NMI address, and parity generators for the address/data received and transmitted on the NMI. Associated with the CSRs are interrupt request levels asserted when a timeout error or a parity error (internal to the NBIA or an NBIB) is detected, and when the NBI is powered up.

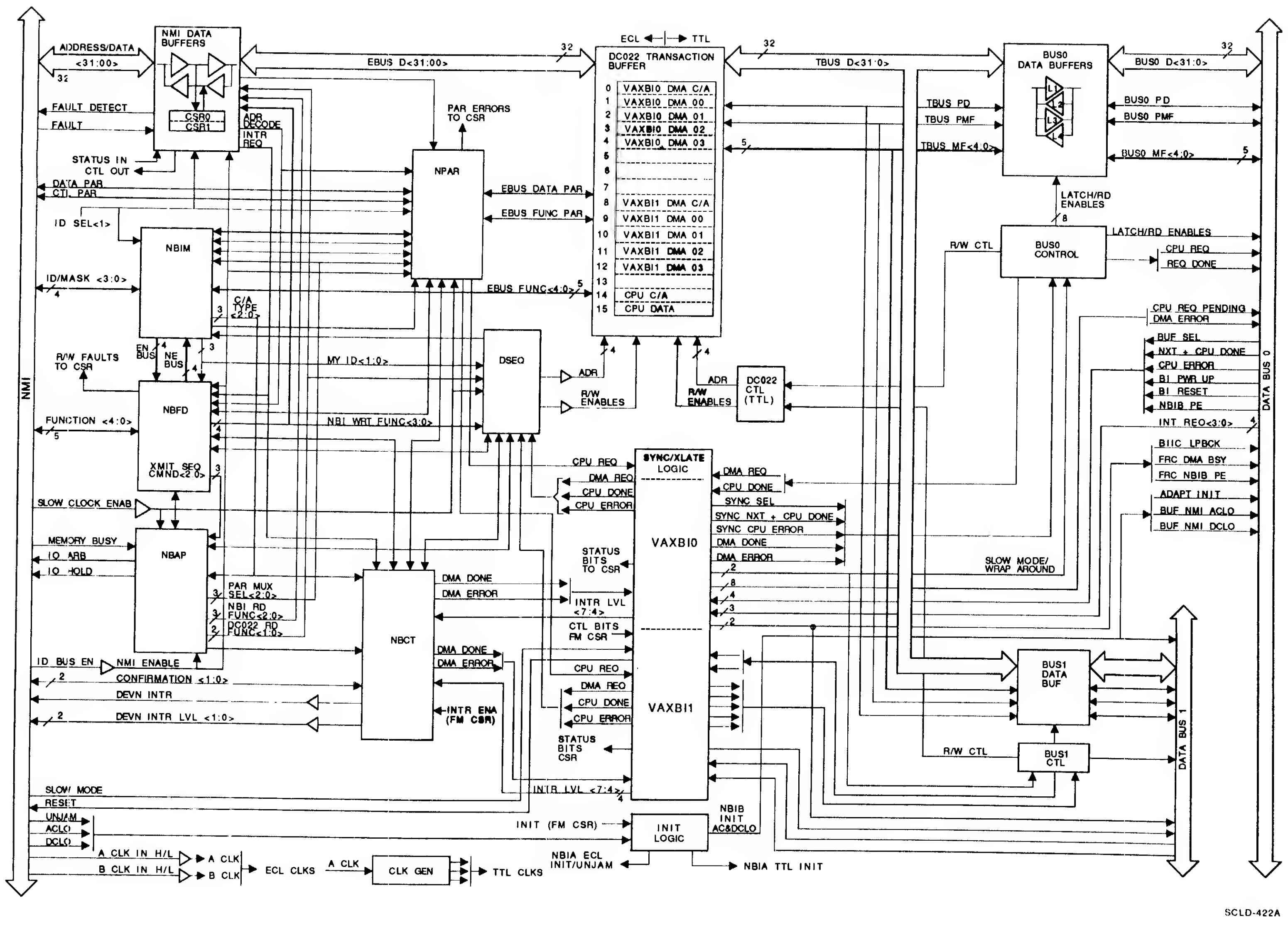


Figure 3-1 NBIA Detailed Block Diagram

3.1.1.2 NPAR MCA -- The NPAR MCA contains the logic for generating the data and control parity bits for information transmitted on the NMI, and for information into the NBIA's own transaction buffer. The logic also checks data and control line parity for the information received on the NMI, and for information read from the transaction buffer. Data parity is generated and checked using the outputs of the parity generators in the NMI data buffer. Control parity is generated and checked using the outputs of parity generators in the NBIM and NBFD MCAs.

In addition to the parity logic, the NPAR MCA contains timeout counters and related (confirmation check) control logic. It also contains CPU read/write request logic.

3.1.1.3 NBIM MCA -- The NBIM MCA is the NBIA's interface to the NMI ID/MASK lines. It also connects to the transaction buffer and to the NBFD MCA (the interface to the NMI FUNCTION lines). The main function of the NBIM MCA is to act as a multiplexer, to pass either a write mask or a read/write command to/from the transaction buffer, depending upon the operation in progress. In addition to multiplexing logic, the NBIM MCA contains parity generators for the ID/mask information received and transmitted on the NMI, and parity generators for the mask or read/write command passed to/from the transaction buffer (over the EBus). The NBIM MCA also contains logic to decode and save the ID on the NMI, and to decode the read/write command passed from the transaction buffer (to the NBFD MCA).

3.1.1.4 NBFD MCA -- The NBFD MCA contains the transmit/receive latches for interfacing to the NMI FUNCTION lines. (The NMI information is gated to/from multiplexing logic in the NBIM MCA as described in Section 3.1.1.3.) However, its main component is the receive sequencer. The sequencer's outputs are the principal NMI interface control signals when information (command/address or data) is being received from the NMI. The NBFD MCA also contains a parity generator for the read/write command (or status) received on the NMI.

3.1.1.5 NBAP MCA -- The NBAP MCA contains the NMI arbitration line logic. It also contains the transmit sequencer. The outputs from this sequencer are the principal NMI interface control signals when information (command/address and data) is being transmitted on the NMI.

3.1.1.6 NBCT MCA -- The NBCT MCA contains the NMI interrupt and confirmation line logic, the error timeout encoder (outputs loaded in CSR0), and DMA response logic. The response logic signals the end of a DMA read/write operation by asserting DMA DONE (or DMA ERROR).

3.1.1.7 DSEQ MCA -- The DSEQ MCA contains a control sequencer that controls access to the DC022 transaction buffer (via its ECL port). A 4-bit DC022 address and read enable are generated. (External discrete logic generates a write enable.) The sequencer stores DMA requests from the NBIB(s), and then controls the initiation (and the retry, if necessary) of the DMA read/write data transfers by the NMI interface. It also monitors and controls the response of the NMI interface to CPU read/write requests.

3.1.1.8 DC022 Transaction Buffer -- The 16-location X 40-bit transaction buffer consists of ten 4-bit DC022 RAM chips connected in parallel. The RAM chips have independent ECL and TTL ports allowing simultaneous asynchronous read/write operations by the NBIA's ECL and TTL interfaces. In each location used (12 are used), 32 bits are reserved for an address or data, 5 bits for a read/write command or write mask/read status information, and 2 bits for data and control parity. (One of the 40 bits is not used.) Basic read/write timing for both RAM ports is shown in Appendix A.

3.1.1.9 Data (Bus) Buffers -- Each of the NBIA's two data buffers, one for each data bus port, consists of ten 8-bit 74F544 transceiver latch chips. The chips are connected to provide two 40-bit buffers (L1 and L3) for information transmitted to an NBIB, and two 40-bit buffers (L2 and L4) for information received from an NBIB. (As for the transaction buffer, only 39 of the available 40 bits of buffering are used.)

3.1.1.10 Data Bus (and Transaction Buffer) Controls -- The main function of the data bus controls, one per data bus port, is to assert the read and write (latch) enables for the associated data bus buffer in the NBIA. (A bus control also asserts some of the enables for the data bus buffer in the associated NBIB.) Each bus control is a control sequencer consisting mainly of an FPLA and four PALS. Each sequencer's present state outputs, used internally to generate the bus control outputs, are also gated to DC022 transaction buffer control logic (two more PALs) that generate the address for the TTL port. Other bus control outputs cause additional (discrete) transaction buffer control logic to generate the DC022 read/write enables.

3.1.2 NBIB Block Diagram

The NBIB (Figure 3-2) consists of the following major logic elements:

- Data bus data buffer
- BCI data buffer
- Parity and translation logic
- Data buffer read/write control
- Length and interrupt control logic
- Master and slave port sequencers
- BIIC
- VAXBI clock driver/receiver

3.1.2.1 Data Bus Data Buffer -- The NBIB's data bus buffer is similar to a data bus buffer in the NBIA except that there are 20 (instead of 10) 74F544 transceiver latch chips. This gives four 40-bit buffers (L5, L7, L9, and L1) for information received by the NBIB, and four 40-bit buffers (L6, L8, L1, and L2) for information transmitted by the NBIB. Again, only 39 of the available 40 bits of buffering are used.

3.1.2.2 BCI Data Buffer -- The BCI data buffer uses 74F373 latch chips. Ten of these chips are configured to provide two 40-bit buffers for BIIC information. (Only 37 bits in each buffer are used.) One buffer (L3/L5) is for information transmitted to the BIIC, and the other (L4/L6) is for information received from the BIIC. In each buffer, one group of latches (L3 or L4) is for the 32 bits of VAXBI address or data transferred. Another group (L5 or L6) is for the four bits of VAXBI command/status/mask (I line) information and the single VAXBI parity bit.

3.1.2.3 Parity and Translation Logic -- When information is transferred between the data bus buffer and BCI data buffer, parity generator and PAL circuits generate (and check) parity for the information, as well as translate the command/status codes. The command/status codes are translated from NMI to VAXBI format, or from VAXBI to NMI format, depending upon the flow of information.

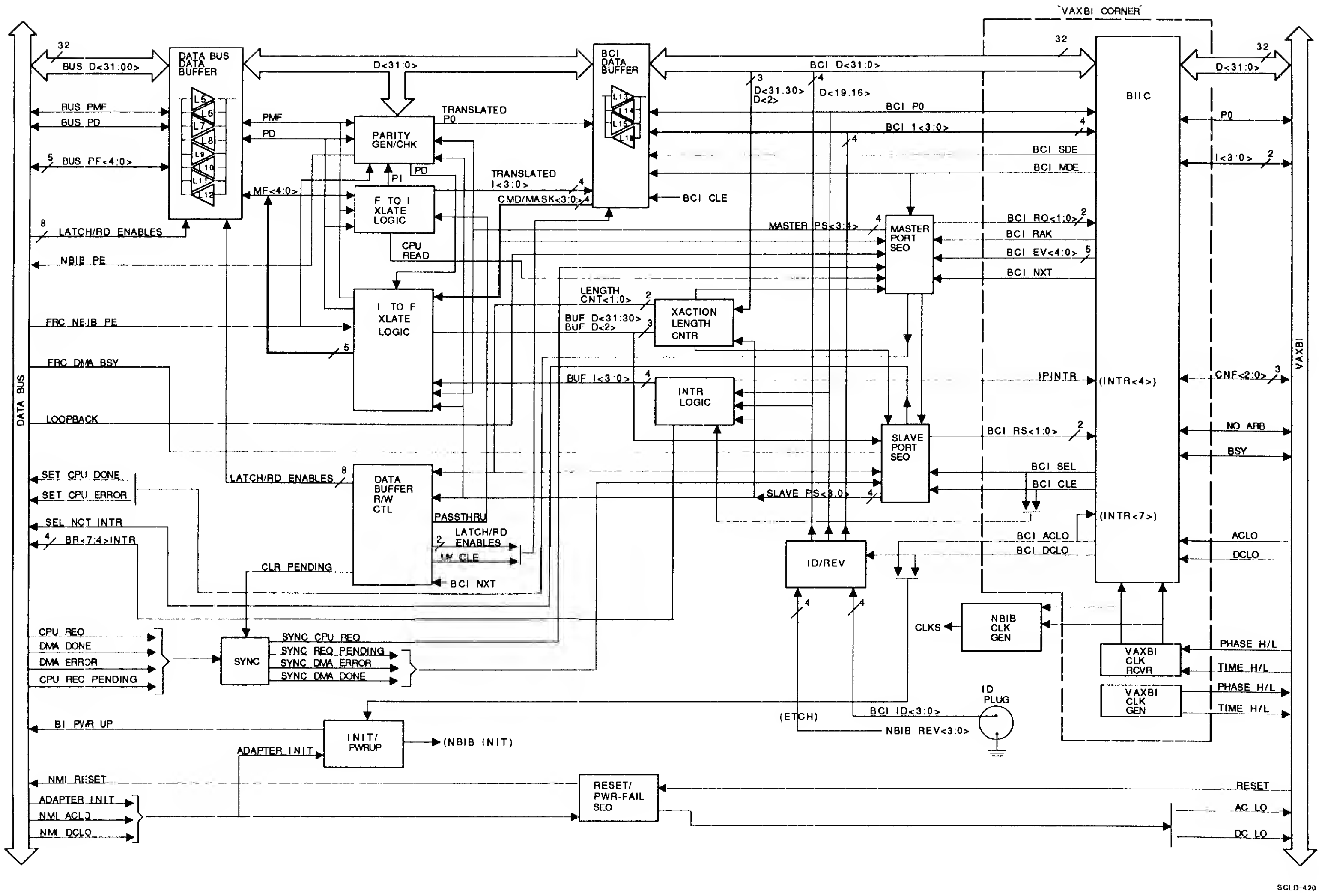


Figure 3-2 NBIB Detailed Block Diagram

3.1.2.4 **Data Buffer Read/Write Control** -- The data buffer read/write control, by asserting the appropriate read and write (latch) enables, causes data to be transferred between the data bus buffer and the BCI data buffer. It consists of three PALs.

3.1.2.5 **Length and Interrupt Control Logic** -- The length control logic and the interrupt control logic store the state of certain VAXBI lines when information is received by the BIIC and transmitted on the BCI. Each consists of a set of latches and a control PAL. The length logic stores the transaction length code (and an address bit) asserted on the data lines at the beginning of a DMA transfer. An internal length counter then signals when the transfer is complete. The interrupt logic stores the interrupt request (BR) levels asserted on the data lines during an INTR transaction. It also stores the state of the I lines, asserting a decoder output when an IPINTR transaction has been received, and it clears the associated (stored) interrupt request during an IDENT transaction.

3.1.2.6 **Master and Slave Port Sequencers** -- The principal control elements in the NBIB are the master and slave port sequencers. Each consists of an FPLA and two PALs. The first sequencer interfaces to the BIIC's master port and controls operation during CPU read/write operations. That is, it initiates the VAXBI read/write or IDENT transactions by the BIIC and then (by means of its present state outputs) controls the transfer information through the NBIB. The second sequencer connects to the BIIC's slave port and controls operation during DMA read/write operations and also during the fielding of interrupt requests. That is, it responds to VAXBI read/write and INTR/IPINTR transactions received by the BIIC, and then (like the master port sequencer) controls the transfer of information through the NBIB by means of its present state outputs.

3.1.2.7 **BIIC** -- The BIIC is a 133-pin ZMOS integrated circuit that serves as a standard, general-purpose interface between the VAXBI and the user interface control logic of a node (the NBIB, in this case.) Its interface to the NBIB logic is called the BCI. The BCI consists of address/data, parity, and information (I) lines that are functionally equivalent to the corresponding VAXBI lines. It also consists of master and slave port (and miscellaneous) control/status signals to control the initiation of, and the response to, VAXBI transactions by the BIIC. BCI signals are defined in Table 3-1.

3.1.2.8 VAXBI Clock Driver/Receiver -- The NBIB generates the clock for the VAXBI it is connected to. As a result, it contains a VAXBI clock driver chip in addition to the standard VAXBI clock receiver contained in all nodes. Both the clock driver and the receiver are custom bipolar integrated circuits. The driver is designed to drive sixteen clock receivers.

Table 3-1 BCI Signals

Signal Line(s)	Number	Description
BCI D<31:00> H	32	Bidirectional tri-state lines used for all address and data transfers.
BCI I<3:0> H	4	Bidirectional tri-state lines used to transfer the command, read status, and write mask. (Refer to Table 2-3.)
BCI PO H	1	Bidirectional line used to generate odd parity for the BCI D and I lines.
BCI RQ<1:0> L	2	Request lines that cause the BIIC to perform a VAXBI transaction, a loopback transaction, or enter diagnostic mode.

BCI RQ <1:0>	Request
0 0	No Request
0 1	VAXBI Transaction Request
1 0	Loopback Request
1 1	Diagnostic Mode

Signal Line(s)	Number	Description
BCI MAB L	1	Master abort line that causes the BIIC to abort the current master port transaction.
BCI RAK L	1	Request acknowledge line asserted by the BIIC to indicate it has initiated a transaction requested by the master port.
BCI NXT L	1	Next line asserted by the BIIC to request that the next data word be asserted on the BCI (for writes), and that the current data asserted on the BCI is valid (for reads).
BCI MDE L	1	Master data enable line asserted by the BIIC to request that a command/-address be asserted on the BCI.

Table 3-1 BCI Signals (Cont)

Signal Line(s)	Number	Description
BCI RS<1:0> L	2	Response lines that specify the confirmation code to be asserted by the BIIC (on the VAXBI CNF lines) in response to command/address and data cycles. (NO ACK code always asserted on CNF lines if BIIC not involved in VAXBI transaction.)

BCI RS <1:0>	Response Code	Resulting CNF Code
0 0	NO ACK	NO ACK
0 1	ACK	ACK, NO ACK
1 0	STALL	STALL, ACK or NO ACK
1 1	RETRY	RETRY, NO ACK

Signal Line(s)	Number	Description
BCI CLE H	1	Command latch enable line asserted by the BIIC to indicate that a VAXBI command/cycle is in progress.
BCI SDE L	1	Slave data enable line asserted by the BIIC to request that read data be asserted on the BCI.
BCI SEL L	1	Select line asserted by the BIIC to indicate it has been selected by a VAXBI transaction.
BCI SC<2:0> L	3	Select code lines asserted by the BIIC when it asserts SEL in order to give detailed selection information. (These lines not used in NBIB.)
BCI AC LO L	1	Buffered and synchronized version of BI AC LO L.
BCI DC LO L	1	Buffered and synchronized version of BI DC LO L.
BCI INT<7:4> L	4	Interrupt request lines that cause the BIIC to generate a VAXBI INTR transaction.
BCI EV <4:0> L	5	Event code lines used to indicate significant events in the BIIC or on the VAXBI.

Table 3-1 BCI Signals (Cont)

Signal Line(s)	Number	Description

BCI EV<4:0>	Mnemonic	Description
00000	NO EVENT	--
00001	MCP	Master Port transaction complete
00010	AKRSD	ACK received for slave read data
00011	BTO	Bus timeout
00100	STP	Selftest passed
00101	RCR	RETRY CNF received (as master)
00110	IRW	Internal register written
00111	ARCR	Advanced RETRY CNF received
01000	NICI	NOACK or illegal CNF received (INTR)
01001	NICIPS	NOACK or illegal CNF received (Force IPINTR/STOP)
01010	AKRE	ACK received for error vector
01011	IAL	IDENT ARB lost
01100	EVS4	External vector selected (BR4)
01101	EVS5	External vector selected (BR5)
01110	EVS6	External vector selected (BR6)
01111	EVS7	External vector selected (BR7)
10000	STO	Stall timeout detected (as slave)
10001	BPS	Bad parity received (as slave)
10010	ICRSD	Illegal CNF received for slave data
10011	BBE	Bus busy error
10100	AKRNE4	ACK received for nonerror vector (BR4)
10101	AKRNE5	ACK received for nonerror vector (BR5)
10110	AKRNE6	ACK received for nonerror vector (BR6)
10111	AKRNE7	ACK received for nonerror vector (BR7)
11000	RDSR	Read data substitute or reserved status received
11001	ICRMC	Illegal CNF received for command
11010	NCRMC	NO ACK CNF received for command
11011	BPR	Bad parity received
11100	ICRMD	Illegal CNF received (data cycle)
11101	RTO	Retry timeout
11110	BPM	Bad parity received (as master)
11111	MTCE	Master transmit check error

3.2 INITIALIZATION/SELFTEST

3.2.1 Basic NBI Initialization

The NBIA and the NBIBs connected to it are initialized at powerup by a programmed NBI INIT (when the ADAPTER INIT control bit is set in the NBIA), and by an UNJAM console command. The UNJAM command does everything a powerup or programmed NBI INIT does except that it does not clear the state indicators in the NBIA's control/status registers. Control bits are cleared.

Two operations initialize just the NBIB. One is when a control bit (NODE RESET) in the BIIC's VAXBI control/status register is set. Another is when a BI RESET signal is received from a connected VAXBI device (a VAXBI to CI adapter, for example). Although BI RESET does not directly initialize the NBIA, it does do it indirectly by causing the NBIA to assert RESET on the NMI. That is, the NMI RESET signal initiates a system bootstrap sequence by the console which, during the sequence, initializes the NBIA and reinitializes the NBIB, by means of a programmed NBI INIT.

Except when the BIIC's NODE RESET control bit is set, NBIB initialization is caused by the assertion of DCLO on the VAXBI. (Setting the BIIC control bit simulates a received DCLO signal; DCLO is not actually asserted on the VAXBI.) Furthermore, the BI DCLO signal, and also BI ACLO, are asserted by the NBIB itself.

During powerup, it is NMI ACLO and DCLO that cause the NBIB to assert BI ACLO and DCLO. (Circuitry in an expander cabinet also asserts BI ACLO and DCLO during powerup.) A programmed NBI INIT, UNJAM command, or BI RESET cause the NBIB to generate a simulated ACLO/DCLO power-fail/powerup sequence (just as if power had failed and then been restored). Because the NBIB normally asserts ACLO and DCLO on the VAXBI when it is initialized, an NBIB initialization also initializes all the connected VAXBI devices. After initialization is complete (both BI AC LO and DCLO have been deserted), the NBIB sets a POWERUP status bit in the NBIA causing it to generate an interrupt request to the CPU.

NBI initialization is summarized in Table 3-2.

Table 3-2 NBI Initialization

Initializing Signal	NBIA	NBIB
NBI INIT (NBIA CSR)	INIT	BI ACLO/BI DCLO sequence (INIT)
NODE RESET (BIIC CSR)	N/A	INIT
NMI UNJAM	INIT (not status bits)	BI ACLO/BI DCLO sequence (INIT)
BI RESET	N/A	BI ACLO/BI DCLO sequence (INIT)
NMI ACLO	N/A	BI ACLO
NMI DCLO	INIT	BI DCLO (INIT)

3.2.2 BIIC Initialization/Selftest

During NBIB initialization, BI DCLO causes the NBIB's BIIC to do the following:

1. Load the node ID in its VAXBI control/status register.
2. Load the NBIB module revision level in its device register.
3. Set the USER PARITY ENABLE bit in its bus error register.

Then, when BI DCLO is deserted, the BIIC performs a selftest. If the test completes successfully, the SELFTEST STATUS bit in the VAXBI control/status register is set, and the two LEDs on the module (visible from the front of the cabinet) are lit.

For the BIIC to load the node ID, revision level, and the USER PARITY ENABLE bit, the NBIB logic must drive nine of the BIIC's BCI interface lines during the time BI DCLO is asserted. The node ID, which is determined by the ID plug inserted in the backplane, is asserted on the four I lines. The revision level, which is specified by jumpers (etch) on the module, is asserted on four of the data lines (<19:16>). To set the USER PARITY ENABLE bit, the NBIB logic deasserts the parity (P0) line. This bit disables VAXBI parity generation by the BIIC when it is transmitting information (command/address or data) on the VAXBI that is supplied by the NBIA. The parity bit (computed in the NBIB) is supplied to the BIIC with the information to be transmitted.

To light the LEDs after successful completion of the BIIC selftest, the NBIB's master port sequencer sets a flip-flop (BCI STPASS) whenever the the BIIC asserts an STPASS (selftest passed) code on its event code lines. The flip-flop output, asserted low, causes +5 V to be applied across the two LEDs (and a resistor) connected in series. Both LEDs are lit because there is no selftest implemented for the NBIB logic. (In some VAXBI nodes, one LED is the BIIC selftest indicator, and the other is the user interface selftest indicator.)

3.2.3 Powerup

The power-up sequence is shown in Figure 3-3. As system power comes on, NMI ACLO and DCLO (generated in the CPU cabinet's power supply) are both initially asserted. The two signals, received by FETs in the NBIA, are asserted on the data bus to each NBIB. Each NBIB then uses the signals to drive FETs connected to the BI ACLO and DCLO lines. Thus, BI ACLO and DCLO are asserted as long as NMI ACLO and DCLO are asserted. They can be asserted longer if the NBIB is in an expander cabinet, which also asserts ACLO and DCLO on the VAXBI.

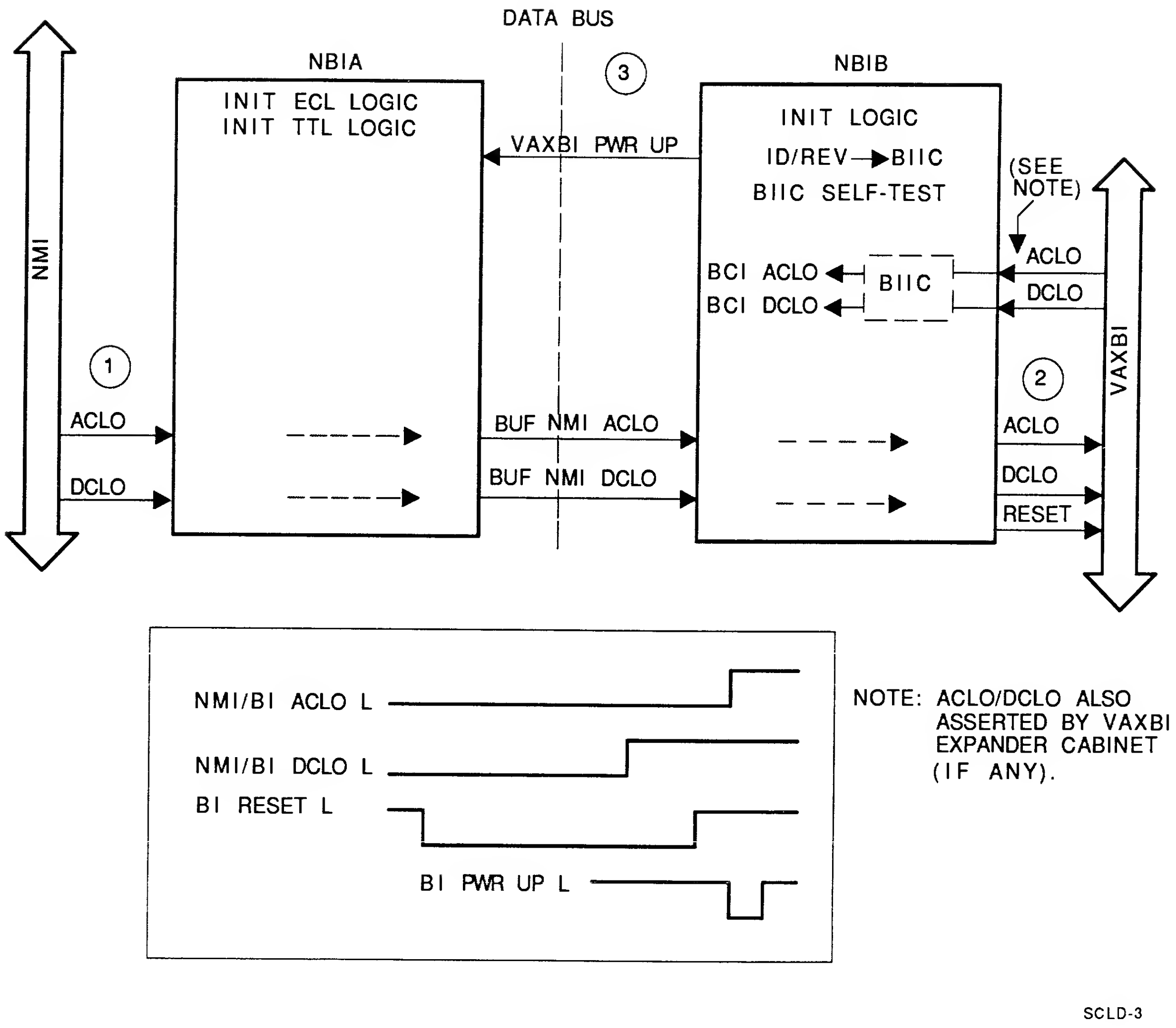


Figure 3-3 NBI Powerup

The ACLO signal causes no initialization in the NBIA or NBIB. However, NMI DCLO initializes both the ECL and TTL logic in the NBIA by asserting NBI INIT (ECL) and NBIA INIT (TTL). Similarly, BI DC LO is received by the BIIC and initializes the NBIB logic by asserting NBIB INIT. The received DCLO signal also causes register bits in the BIIC to be initialized (Section 3.2.2).

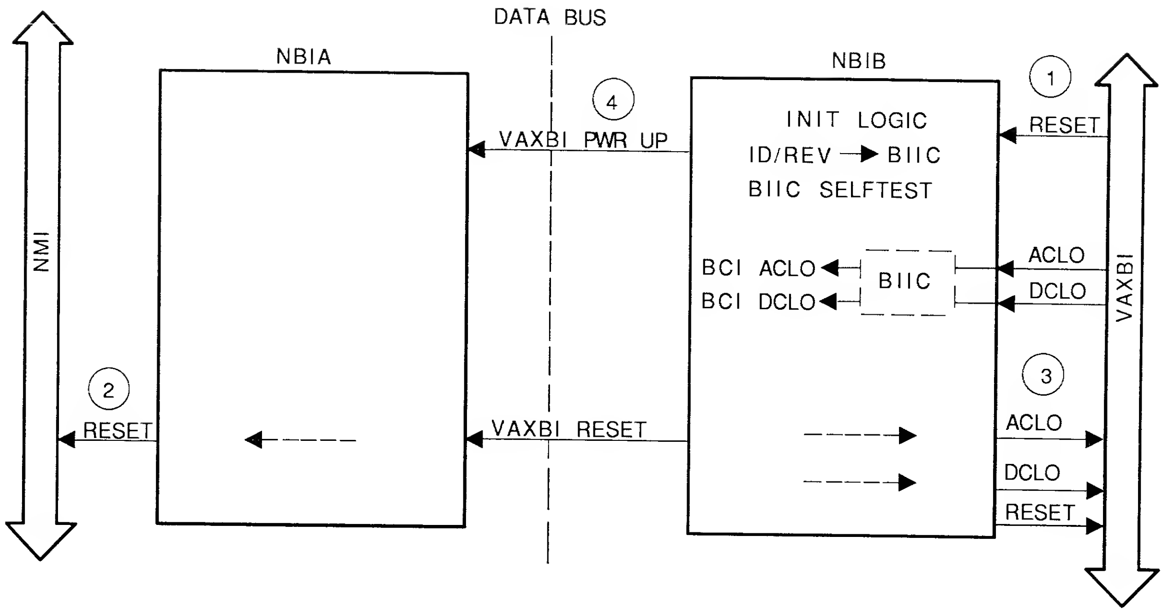
The BIIC receives BI ACLO in addition to BI DCLO, and both signals (asserted by the BIIC as BCI ACLO and DCLO) connect to the NBIB's reset/power-up sequencer. (The signals are first synchronized to the sequencer's clock.) During powerup, the assertion of both ACLO and DCLO cause the sequencer to also assert RESET on the VAXBI. The BI RESET signal is asserted while DCLO is asserted and for 100 microseconds after it is deserted. This initiates a "cold start" initialization sequence in the VAXBI devices. BI RESET causes no action in the NBIB during powerup, but the deassertion of DCLO (as always) causes the NBIB's BIIC to perform a selftest (Section 3.2.2).

When BI ACLO is deserted ending the powerup sequence, the NBIB asserts BI PWR UP on the data bus to the NBIA. The corresponding PWR UP status bit is then set in the NBIA's control/status register (in CSR1) causing a CPU interrupt request. There are two PWR UP status bits, one for each VAXBI (each NBIB).

3.2.4 NBI INIT/UNJAM

A programmed NBI INIT causes the NBIA to be initialized just as if NMI DCLO had been asserted. Setting the ADAPTOR INIT control bit in the NBIA's control/status register (in CSR1) asserts both NBI INIT and NBIA INIT which initializes both the ECL and TTL logic. An NMI UNJAM signal also asserts NBIA INIT to initialize the TTL logic. However, NBI UNJAM (not NBI INIT) is asserted, and not all the ECL logic is initialized. The logic left in its current state are the status indicators in the NBIA's control/status registers (CSR0 and CSR1).

Both a programmed NBI INIT and UNJAM operation assert ADAPT INIT on the data bus to each NBIB. Refer to Figure 3-4. This signal causes the reset/power-up sequencer in each NBIB to generate an ACLO/DCLO power-fail/power-up (system reset) sequence on the VAXBI. First, BI RESET is asserted. Also, BI ACLO and then BI DCLO are asserted simulating a power-fail condition. This is followed by a simulated power-up sequence with the sequencer first deasserting BI DCLO, then BI RESET, and finally BI ACLO. As occurs during an actual power-up sequence (Section 3.2.3), the simulated powerup initializes the NBIB and all the connected VAXBI devices. During the NBIB initialization, the BIIC registers are initialized and a BIIC selftest is initiated (Section 3.2.2). Also, at the end of the power-up sequence, the PWR UP status bit is set in the NBIA causing a CPU interrupt request.



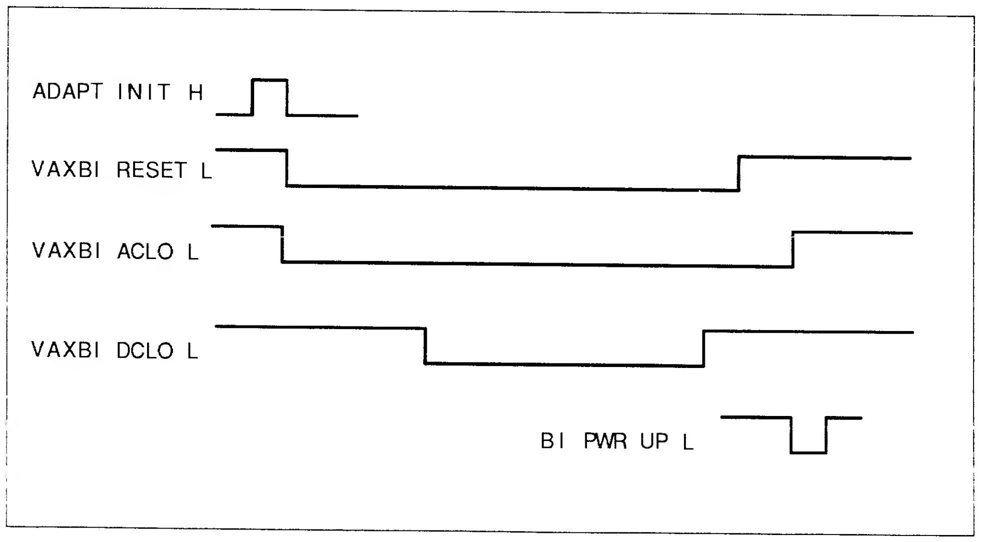
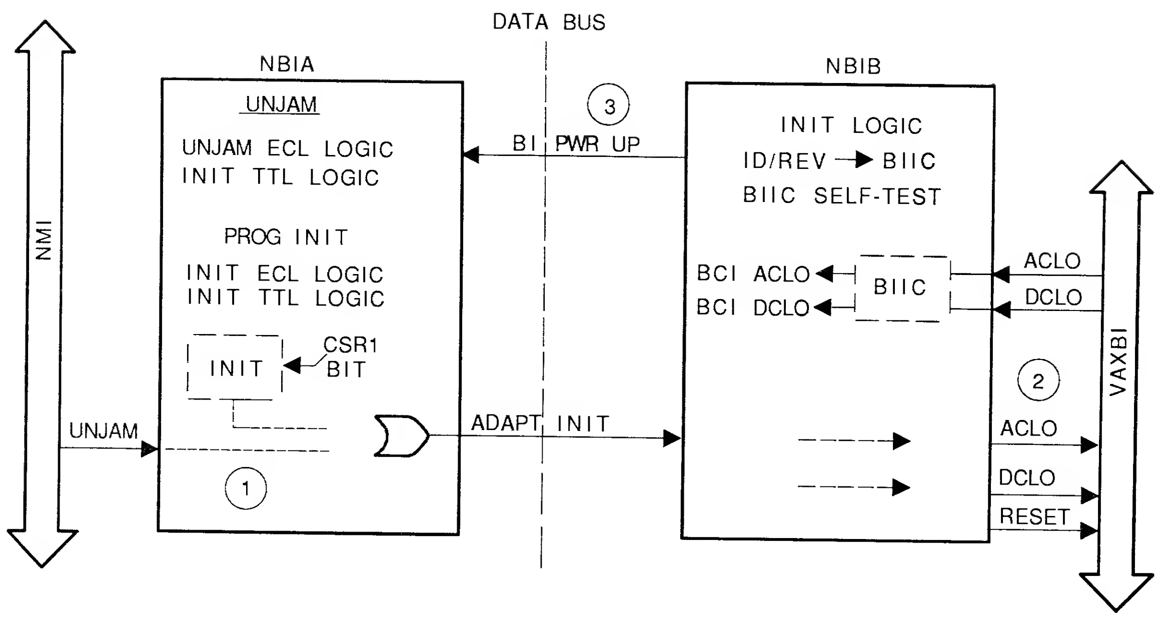


Figure 3-4 Reset by VAXBI Node

3.2.5 RESET (By Connected VAXBI Device)

A remote processor may bootstrap the system by asserting BI RESET via its VAXBI (to CI or NI) adapter node. When BI RESET is received by the NBIB, its reset/power-up sequencer asserts a reset signal on the data bus to the NBIA. (Refer to Figure 3-5.) In the NBIA, the signal asserts NMI RESET causing the console to stop the CPU(s) and bootstrap the system. The system bootstrap is not initiated until NMI RESET is deserted.

After asserting the reset signal to the NBIA, the NBIB's reset/power-up sequencer generates an ACLO/DCLO power-fail/power-up sequence on the VAXBI. This sequence, which is also generated by a programmed NBI INIT and an UNJAM console command (refer to Section 3.2.4), initializes the NBIB and the connected VAXBI devices before the bootstrap begins.



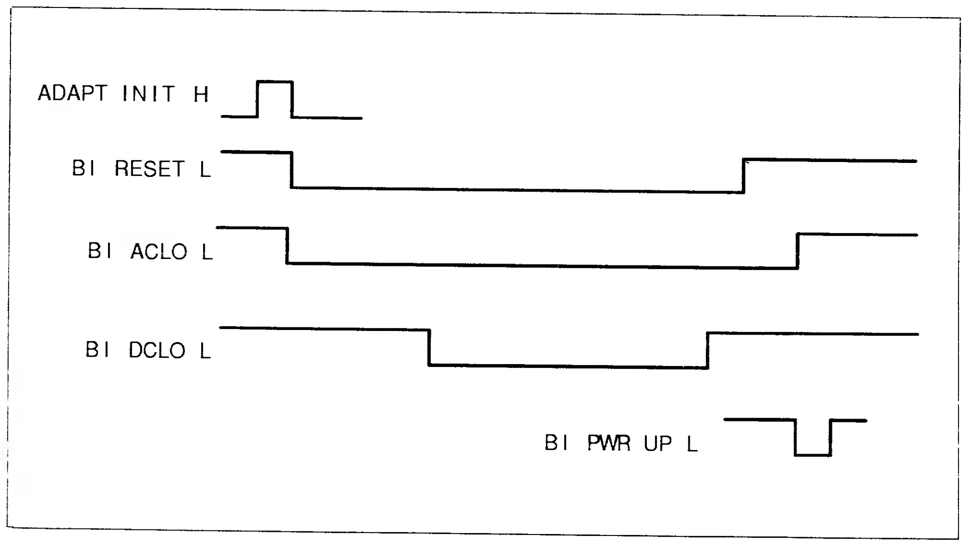


Figure 3-5 UNJAM/Programmed NBI INIT

3.3 CPU READ/WRITE OPERATIONS

The CPU can access the following registers by means of NMI read/write transactions to the NBIA. The transactions are longword data transfers.

1. Read and write the NBIA's control/status registers (CSR0 or 1).

2. Read and write registers in the VAXBI devices connected to an NBIB. To do this, the NBIB must initiate a VAXBI read/write transaction in response to the NMI read/write transaction. The registers accessed may be in any of the 16 nodes connected to a VAXBI, and includes the registers in the NBIB's own BIIC.

3. Read interrupt vectors from the NBIA's four interrupt vector registers (BR4VR through BR7VR). To read VAXBI device vectors, the NBIB must initiate a VAXBI IDENT transaction in response to the NMI read transaction. Reading the NBI interrupt vector (from BR4VR) requires no VAXBI transaction.

CPU read/write operations are summarized in Table 3-3. As shown, the NBIA responds to NMI write masked and write unlock masked transactions in addition to standard NMI write transactions. It also responds to NMI read interlocked transactions as well as standard NMI read transactions. When a VAXBI register is addressed, the NBIB generates the equivalent transaction (standard, masked, interlocked, etc.) on the VAXBI. When an NBIA register is addressed, the NBIA (which does not support internal masked or interlocked operations) executes all NMI transaction types as standard reads or writes.

Table 3-3 CPU Read/Write Summary

NMI Transaction (Note 1)	Address	Interrupt Pending NBI (BR4)	Interrupt Pending BRx	In Progress Loc Rd	In Progress BI R/W BI Vec DMA Wrt	NBI Response	Resulting VAXBI Transaction (Note 1)	See Note
WL/WLM/WLUM	CSRx	-	-	No	-	ACK, Write CSRx		
	CSRx	-	-	Yes	-	BSY		
WL/WLM/WLUM	RSVDx	-	-	-	-	ACK, No write		
WL/WLM/WLUM	BRxVR	-	-	-	-	ACK, No write		
WL/WLM/WLUM	VAXBI	-	-	-	No	ACK, Write BI reg.	WR/WMCI/UWMCI	
	VAXBI	-	-	-	Yes	BSY		
RL/RLI	CSRx	-	-	No	-	ACK, Rtrn CSR data		
	CSRx	-	-	Yes	-	BSY		
RL/RLI	RSVDx	-	-	No	-	ACK, Rtrn zeros		
	RSVDx	-	-	Yes	-	BSY		
RL/RLI	BR4VR	Yes	-	No	-	ACK, Rtrn NBI vect		
	BR4VR	Yes	-	Yes	-	BSY		
	BRxVR	No	Yes	-	No	ACK, Rtrn BI vect	IDENT	2,3
	BRxVR	No	Yes	-	Yes	BSY		
	BRxVR	-	No	No	-	ACK, Rtrn zeros		3
	BRxVR	-	No	Yes	-	BSY		
RL/RLI	VAXBI	-	-	-	No	ACK, Rtrn BI data	RD/IRCI	2
	VAXBI	-	-	-	Yes	BSY		

NOTES:

1. Transaction types.

NMI Transaction	Resulting VAXBI Transaction (Length = L)
WL = Write Longword	WR = Write
WLM = Write Longword Masked	WMCI= Write Masked Cache Intent
WLUM = Write Longword Unlock Masked	UWMCI = Unlock Write Make Cache Intent
RL = Read Longword	RD = Read
RLI = Read Longword Interlocked	IRCI = Interlocked Read Cache Intent

2. Zeros returned on NMI if error detected in reading VAXBI data/vector. NBI's BIIC interrupts CPU to flag error condition.

3. The return of a zero vector indicates interrupt request has been deserted by VAXBI device (or error detected in reading VAXBI vector). Causes passive release (NAP) by CPU interrupt handler.

3.3.1 NMI Address Decoding and Translation

NMI address decoding by an NBIA during CPU read/write operations is shown in Figure 3-6. Also shown is the translation of the NMI address to a VAXBI address when a VAXBI device register is accessed.

An NBIA is selected by a CPU read/write when NMI address bits <29:27> are equal to 100, and address bit <26> matches the NBIA's I/O SEL level. This signal, hardwired on the backplane, defines the NBIA as either NBIA0 or NBIA1.

The remaining address bits are decoded as follows. Bit <25> specifies the VAXBI and thus the NBIB selected. Bits <24:00> determine the register selected. All register addresses are decoded as VAXBI register addresses and passed to the selected NBIB except for one block of addresses within the node private space for VAXBI0. This address block, most of which is not allocated, contains the NMI nexus registers in the NBIA. The NMI nexus registers are the two control/status registers, the two reserved (unused) registers, and the four interrupt vector registers.

When an NMI address is passed to an NBIB and its associated VAXBI, the bits selecting the NBIA and NBIB (bits <26:25>) are forced to zeros. (VAXBI register addresses must fall within the 32 Mb of I/O space allocated to each VAXBI.) No other bits are modified. This includes bits <31:30>, which are not used for addressing purposes on the NMI. However, these bits are used on the VAXBI to specify the transaction length, and the CPU asserts a value of 01 (the length code for a longword transfer on the VAXBI) during the NMI command/address cycle.

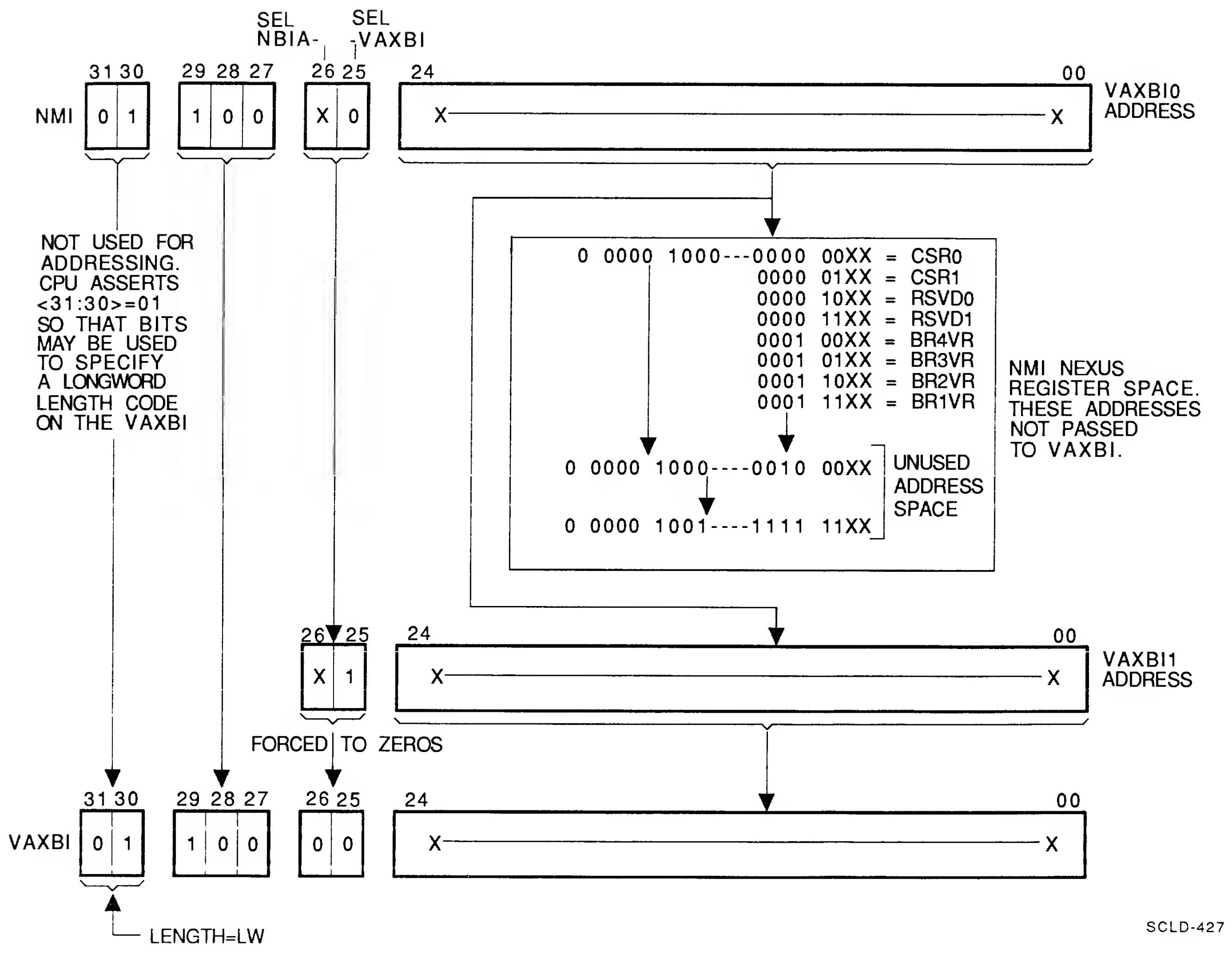


Figure 3-6 NMI Address Decoding and Translation

3.3.2 Local Read/Write Operations

The following are local read/write operations. That is, all address NMI nexus registers in the NBIA and no transfer of data to/from the VAXBI is required.

1. The reading and writing of the CSRs.
2. The reading of the NBI interrupt vector (from BR4VR).
3. The reading of an interrupt vector register (BRxVR) when there is no corresponding interrupt request (BRx). The NBIA returns zeros to the CPU. This condition occurs when a VAXBI or NBI interrupt request is deserted before it can be serviced by the CPU. The return of zeros causes a passive release (NAP) by the CPU's interrupt handler.
4. The reading or writing of the two unused (reserved) NMI nexus registers (RSVDx). The NBIA acknowledges (ACKs) the NMI transaction but writes no data and returns zeros as read data. This is done so that there will be no discontinuities when the NMI nexus registers are read or written as a block. For a similar reason, write transactions addressing the read-only BRxVRs are acknowledged even though no data is written.

A condition exists when a local read/write by the CPU is not acknowledged by the NBIA, at least temporarily. A busy response is returned to the CPU instead. This occurs when a local read has been issued previously and the NBIA has not yet been able to return the read data. (The NBIA must request and win the NMI before it can return read data.) The busy response prevents a local write from modifying a register that is being read. It also prevents stacked local reads, which are not supported by the NBIA. The busy response is not generated for local writes directed to the read-only vector registers or the reserved registers. This is because no data is written by these transactions.

A description of NBIA operation during local reads and writes follows.

3.3.2.1 Command/Address Cycle -- During NMI command/address cycles (Figure 3-7), the CPU asserts the register address on the NMI ADDRESS/DATA lines, its ID on the NMI ID/MASK lines, and the read/write command on the NMI FUNC lines. The CPU also asserts even parity on the NMI's data and control parity lines.

In the NBIA, the address is partially decoded by the NMI data buffer (the ND MCAs) causing the receive sequencer in the NBFD MCA to advance state and start a local read/write if the following conditions are true.

1. The NBIA is addressed (address bit <26> = I/O SEL).

2. An NMI nexus register is addressed, but not when BRxVR is addressed to read a VAXBI device vector (BRx interrupt request from VAXBI device pending). An exception is when BR4VR is read and there is both a a corresponding VAXBI device interrupt request and an NBI interrupt request pending (both BR4 requests). In this case, the NBI interrupt request has a higher priority and a local read is started to return the NBI vector.

3. The read/write command received on the FUNC lines is a read/write longword command.

4. CSR BUSY is not set (unless a read-only vector register or a reserved register is being written). CSR BUSY = 0 indicates a local read is not already in progress.

5. No NMI parity error was detected by the NPAR MCA.

To start a local read/write, the NBFD's receive sequencer asserts SEND ACK to the NBCT MCA causing an ACK response to be returned to the CPU on the NMI CNF lines. The ACK response is asserted during the second bus cycle following the command/address cycle.

For a local read, the receive sequencer also does the following before returning to its idle state.

1. Sets CSR BUSY to indicate a local read is in progress.

2. Asserts an NBI write function (SAVE ID) that causes the NBIM MCA to store the CPU's ID received on the NMI CNF lines.

3. Asserts a transmit sequence command, latched by the NBAP MCA, that indicates what local register data (or if zero data) is to be returned to the CPU. The command causes the NBAP's transmit sequencer to start the read data cycle necessary to return the local read data to the CPU. The read data cycle is discussed in Section 3.3.2.3.

For a local write, the receive sequencer simply waits for the write data cycle following the command/address cycle.

If a local read is in progress (CSR BUSY = 1) when the command/address is received, and the NBIA must return a busy (BSY) response to the CPU, the NBFD's receive sequencer asserts SEND BUSY instead of SEND ACK. This causes the NBCT MCA to transmit the BSY response (instead of the ACK response) on the NMI CNF lines. The receive sequencer returns to the idle state after asserting SEND BUSY. No other NBIA operations take place. Like the ACK response, the BSY response is asserted on the CNF lines two cycles after the command/address cycle.

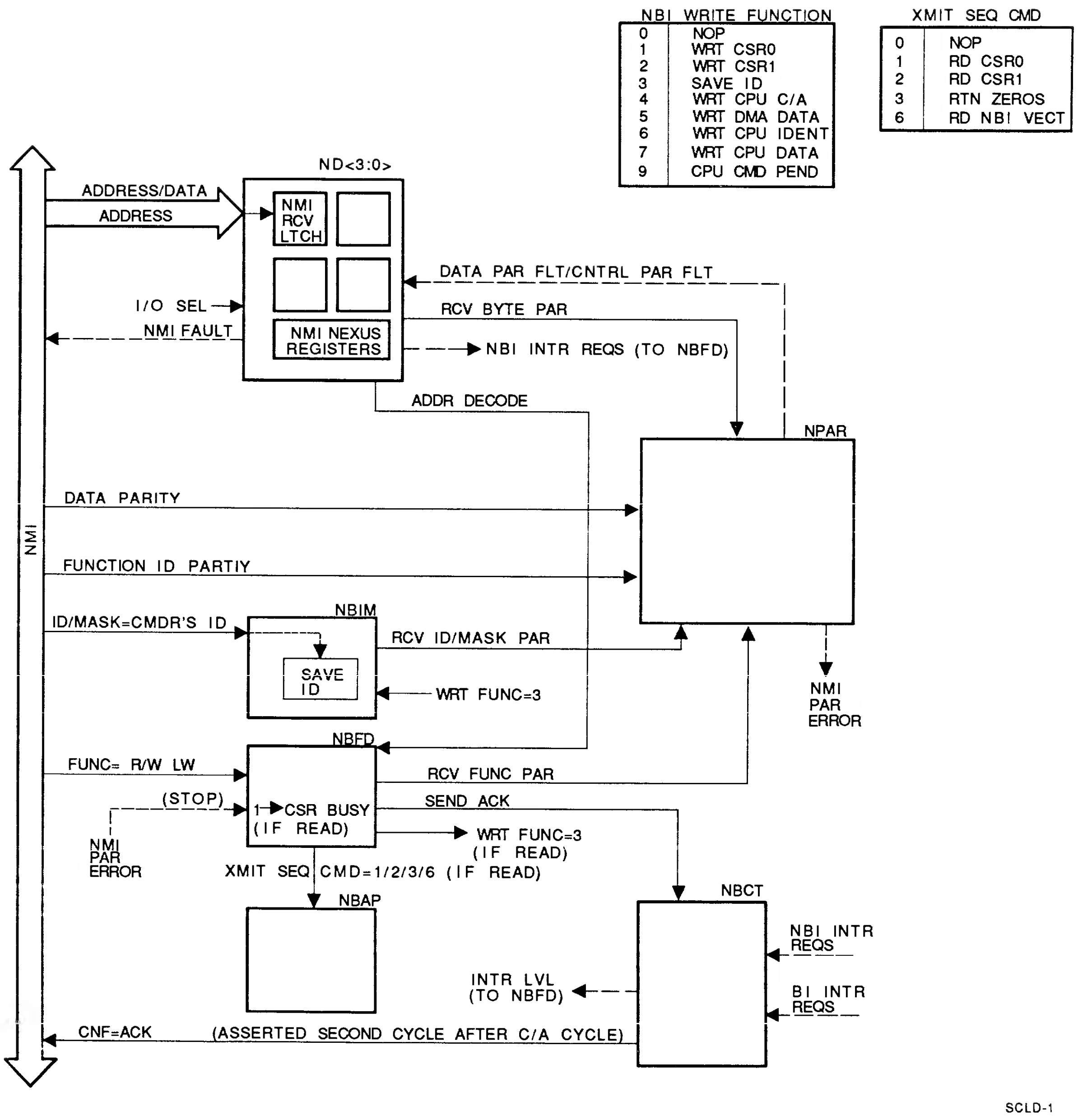


Figure 3-7 Local Read/Write Command/Address Cycle

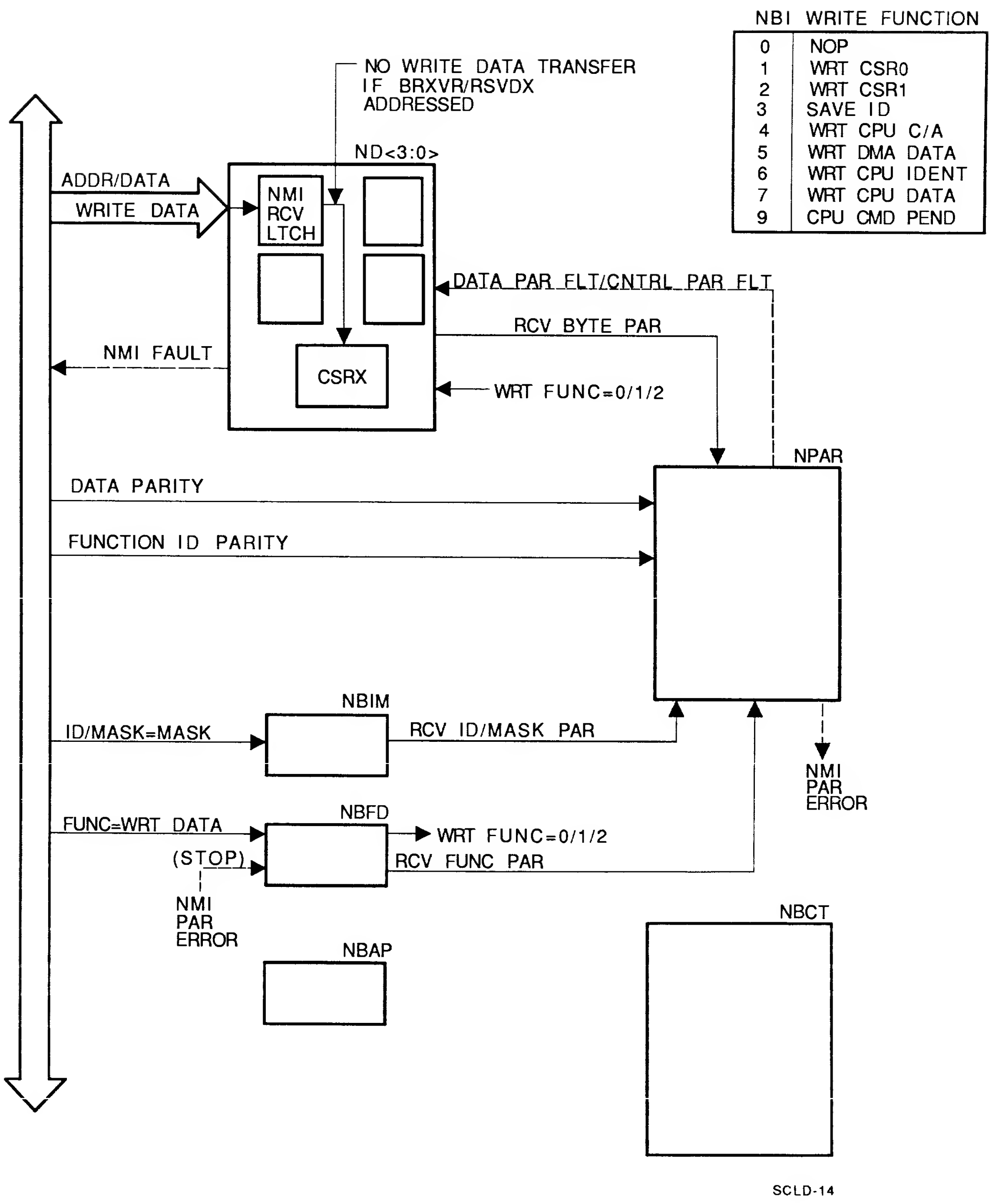
3.3.2.2 Write Data Cycle -- During an NMI write data cycle (Figure 3-8), the CPU asserts the write data on the NMI ADDRESS/DATA lines, mask data (if any) on the NMI ID/MASK lines, and a write data function on the NMI FUNC lines. The CPU also asserts even parity on the NMI's data and control parity lines.

In the NBIA, any mask data is ignored during local write operations. However, the received write data function identifies the NMI data as write data causing the NBFD MCA's receive sequencer (which is in a wait state) to do either of the following:

1. Generate an (nonzero) NBI write function and go to the idle state. The NBI write function causes the NMI data buffer to strobe the write data from the NMI ADDRESS/DATA lines into one of the CSRs. These registers are the only NMI nexus registers that can be written.

2. Just go to the idle state (NBI write function equals 0). This occurs when an NMI nexus register other than a CSR has been addressed and no data is strobed from the NMI ADDRESS/DATA lines.

The return of the receive sequencer to its idle state ends the local write operation in the NBIA.



NBI WRITE FUNCTION

0	NOP
1	WRT CSR0
2	WRT CSR1
3	SAVE ID
4	WRT CPU C/A
5	WRT DMA DATA
6	WRT CPU IDENT
7	WRT CPU DATA
9	CPU CMD PEND

Figure 3-8 Local Write Data Cycle

3.3.2.3 Return Data Cycle -- To return local read data (Figure 3-9), and after latching the transmit sequence command asserted by the NBFD MCA during the command/address cycle, the transmit sequencer in the NBAP MCA does the following:

1. Asserts I/O ARB on the NMI to request use of the bus.

2. Asserts an NBI read function equal to the latched transmit sequence command. The NBI read function value determines what NMI nexus register data (or if zero data) will be returned to the CPU by the NMI data buffer.

3. Asserts RTRN RD DATA. This signal specifies that a RTN DATA function will be returned to the CPU by the NBFD MCA.

Nothing else occurs until the bus is granted to the NBIA by the NMI arbitration logic in the CPU. When the bus is granted (which may be immediately or after a number of bus cycles), I/O BUS EN from the bus arbitration logic asserts NMI ENABLE in the NBIA. The following then takes place.

1. The NMI data buffer transmits the register data (or zero data) that was selected by the read function on the NMI ADDRESS/DATA lines.

2. The NBIM MCA transmits the CPU's ID on the NMI ID/MASK lines. The ID was saved during the command/address cycle. It is selected for transmission by the NBAP MCA, which asserts a PAR MUX SEL code that is equal to 0 (its default value) during the local read cycle.

3. The NBFD MCA transmits a RTN DATA function on the NMI FUNC lines.

Following transmission of the read data and ID on the NMI, the NBIA relinquishes the bus (never asserts I/O HOLD) ending the NBIA local read operation.

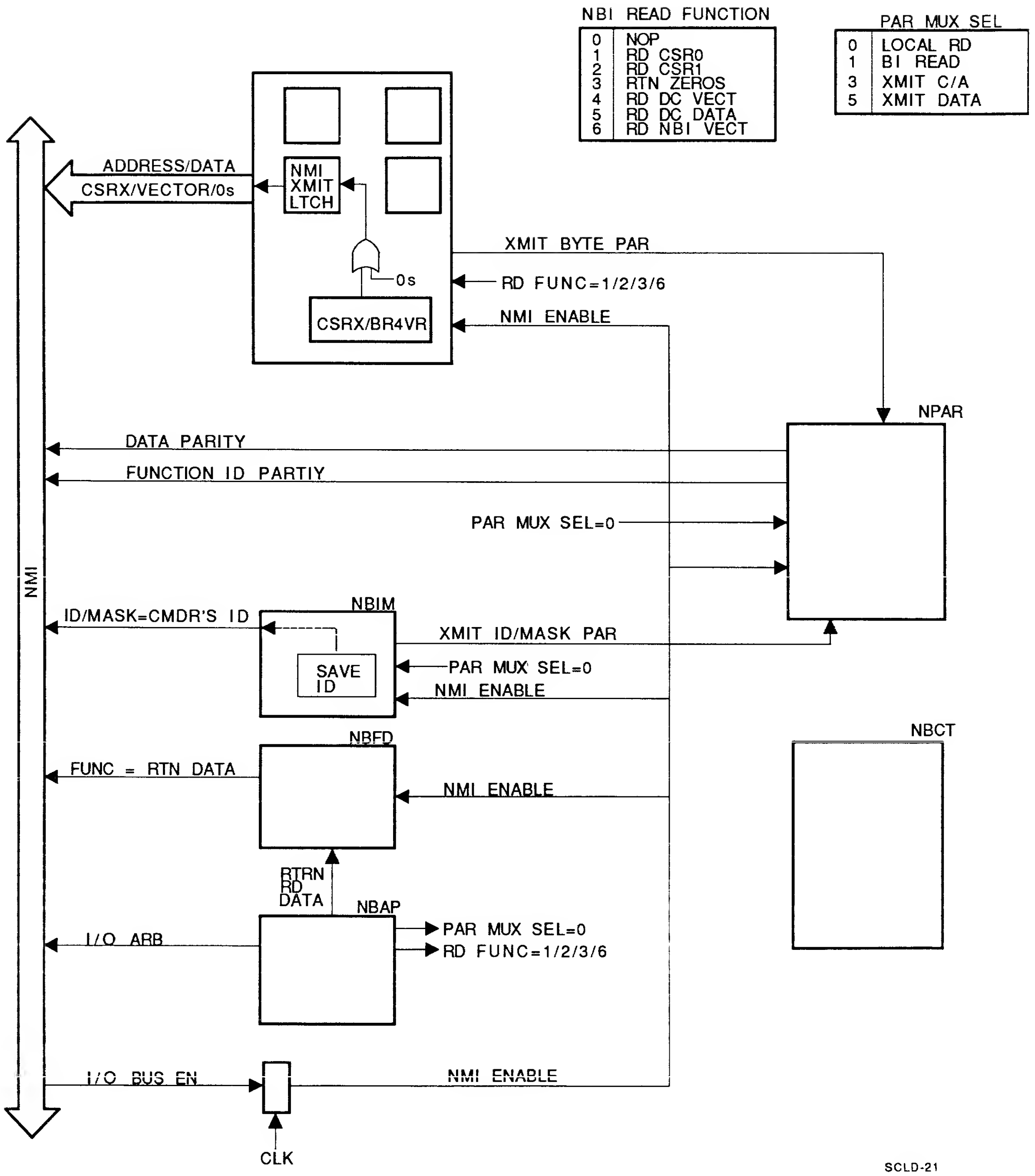


Figure 3-9 Local Read Data Cycle

3.3.2.4 Parity Generation and Checking -- Parity generation and checking during command/address and write data cycles is the same. The NMI data buffer generates an even parity bit for each of the four bytes of address/write data received from the NMI. Similarly, the NBIM MCA generates an even parity bit for the received ID/MASK information, and the NBFD MCA generates an even parity bit for the received NMI function. The NPAR MCA then does the following:

1. Completes the computation of data line parity using the 4-byte parity bits, and compares the result with the data parity bit (DATA PARITY) received from the NMI.

2. Completes the computation of control line parity using both the ID/MASK and function parity bits, and compares the result with the control line parity bit (FUNCTION ID PARITY) received from the NMI.

If bad (odd) NMI data or control parity is detected, the NPAR MCA asserts DATA PAR FLT or CNTRL PAR FLT, which causes the NMI data buffer (the ND3 MCA) to set the corresponding parity error flag in CSR0. This, in turn, asserts the NBI's FAULT DETECT line on the NMI, which causes an interrupt request to be generated in the CPU. (Because all NMI nexus check NMI parity during every bus cycle, all could be asserting their NMI FAULT DETECT line.) The NPAR MCA also asserts NMI PAR ERROR for either type of error. This signal connects to the NBFD MCA, stopping the receive sequencer. As a result, a parity error prevents the NBIA from acknowledging an NMI transaction with a bad command/address. Also, if the command address is good but write data is bad, the transaction is acknowledged but bad data is not written into a register.

During return read data cycles, the NPAR MCA must generate and transmit (not receive and check) the NMI data and control line even parity bits. The NMI data buffer again generates and sends even byte parity bits to the NPAR MCA, but the parity bits are for the NMI data it is transmitting and not receiving. Also, the NBIM MCA generates and sends a parity bit to the NPAR MCA, but the parity bit is now for the ID it is transmitting on the NMI. The NBFD MCA does not generate a parity bit during the return data cycle, however. This is because it always transmits a RTN DATA function on the NMI FUNC lines, which means the NPAR MCA only needs the ID parity bit to generate NMI control parity.

For diagnostic purposes, a control bit (FORCE NBIA PE) in CSR0 can be set that causes the NPAR MCA to generate bad (odd) parity for both the NMI data and control lines when the NBIA returns read data.

3.3.3 VAXBI Read/Write (and IDENT) Operations

A CPU read/write to a VAXBI device register requires first that the NMI command/address be transferred from the NBIA to the associated NBIB so that a VAXBI read/write transaction can be started. For a CPU write, this is followed by a transfer of the NMI write data to the NBIB where is transmitted on the VAXBI (during the VAXBI write transaction). For a CPU read, the VAXBI data read by the NBIB (during the VAXBI read transaction) is passed back to the NBIA where it is returned to the CPU over the NMI.

Another CPU read operation requiring a VAXBI transaction is the reading of a VAXBI device's interrupt vector. The NMI transaction reading the NBIA's vector register (BRxVR) causes the NBIA to generate and transfer an IDENT command/address to the NBIB. The NBIB then does the VAXBI IDENT transaction to collect the vector from the device. Following this, and similar to a VAXBI read operation, the NBIB passes the vector back to the NBIA where it is returned to the CPU over the NMI.

The transfer of command/address and data between the NMI and VAXBI is shown in Figure 3-10.

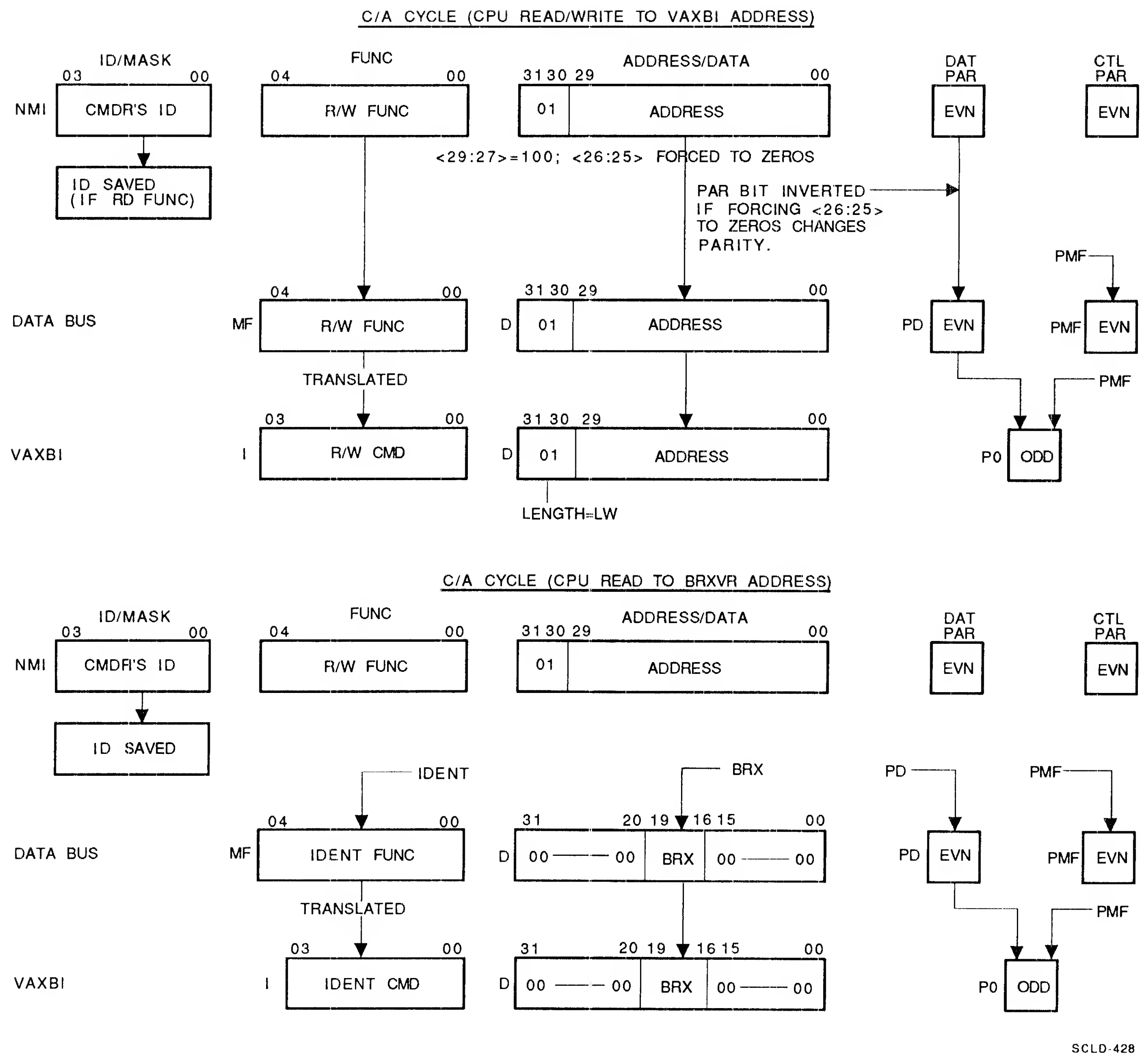


Figure 3-10 Basic Information Flow Between NMI and VAXBI (Sheet 1 of 2)

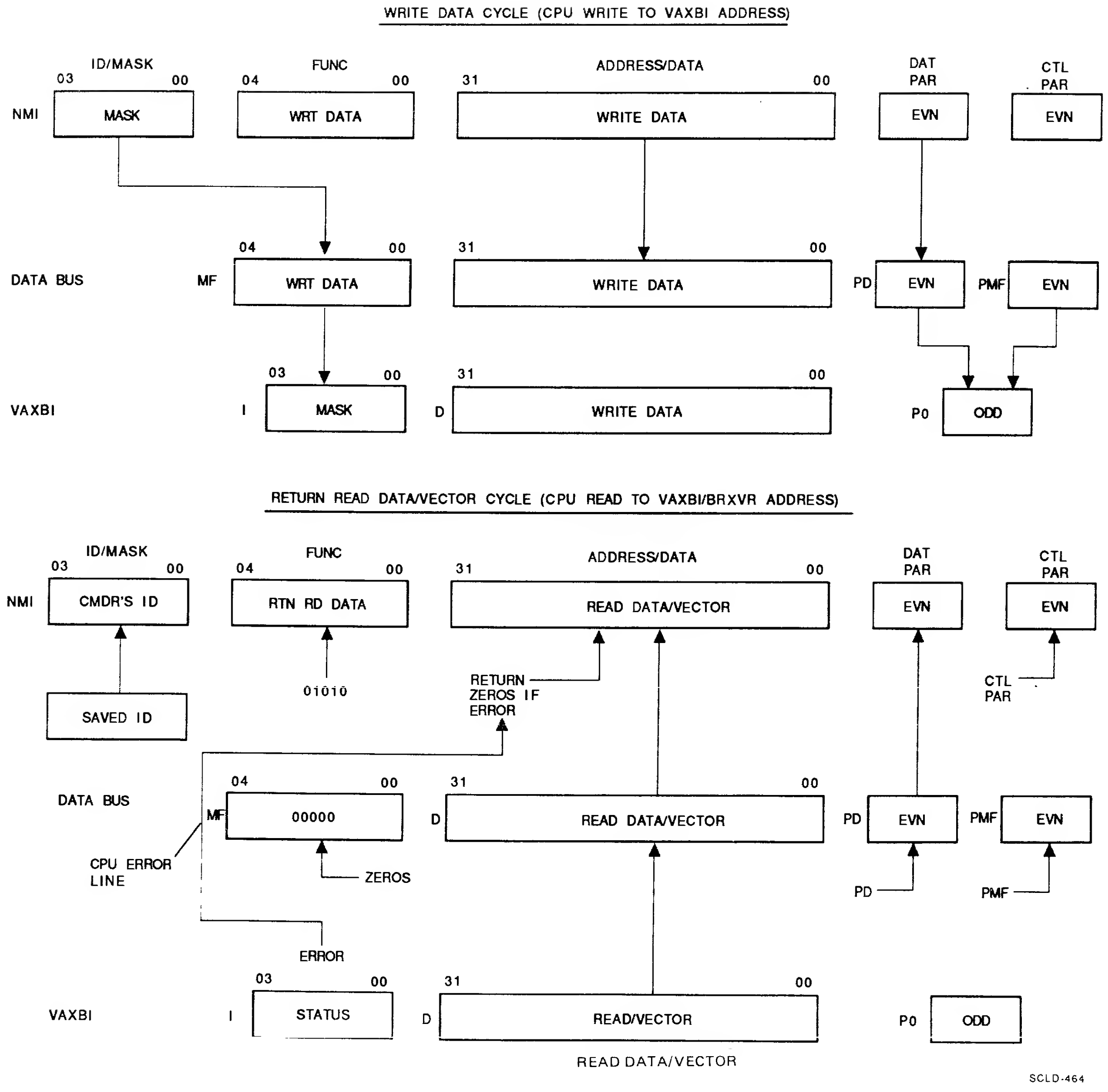


Figure 3-10 Basic Information Flow Between NMI and VAXBI During CPU Read/Write Operations (Sheet 2 of 2)

3.3.3.1 Command/Address Transfer -- The transfer of command/-address information from the NMI to the VAXBI (Figure 3-11) consists of four basic operations.

1. The NBIA loads the NMI command/address (or the generated IDENT command/address) into its DC022 transaction buffer.

2. The NBIA reads the command/address from its transaction buffer and loads it into the NBIB's data bus buffer (over the interconnecting data bus).

3. The NBIB requests a VAXBI transaction by its BIIC, and it reads the command/address from its data bus buffer into its BCI data buffer.

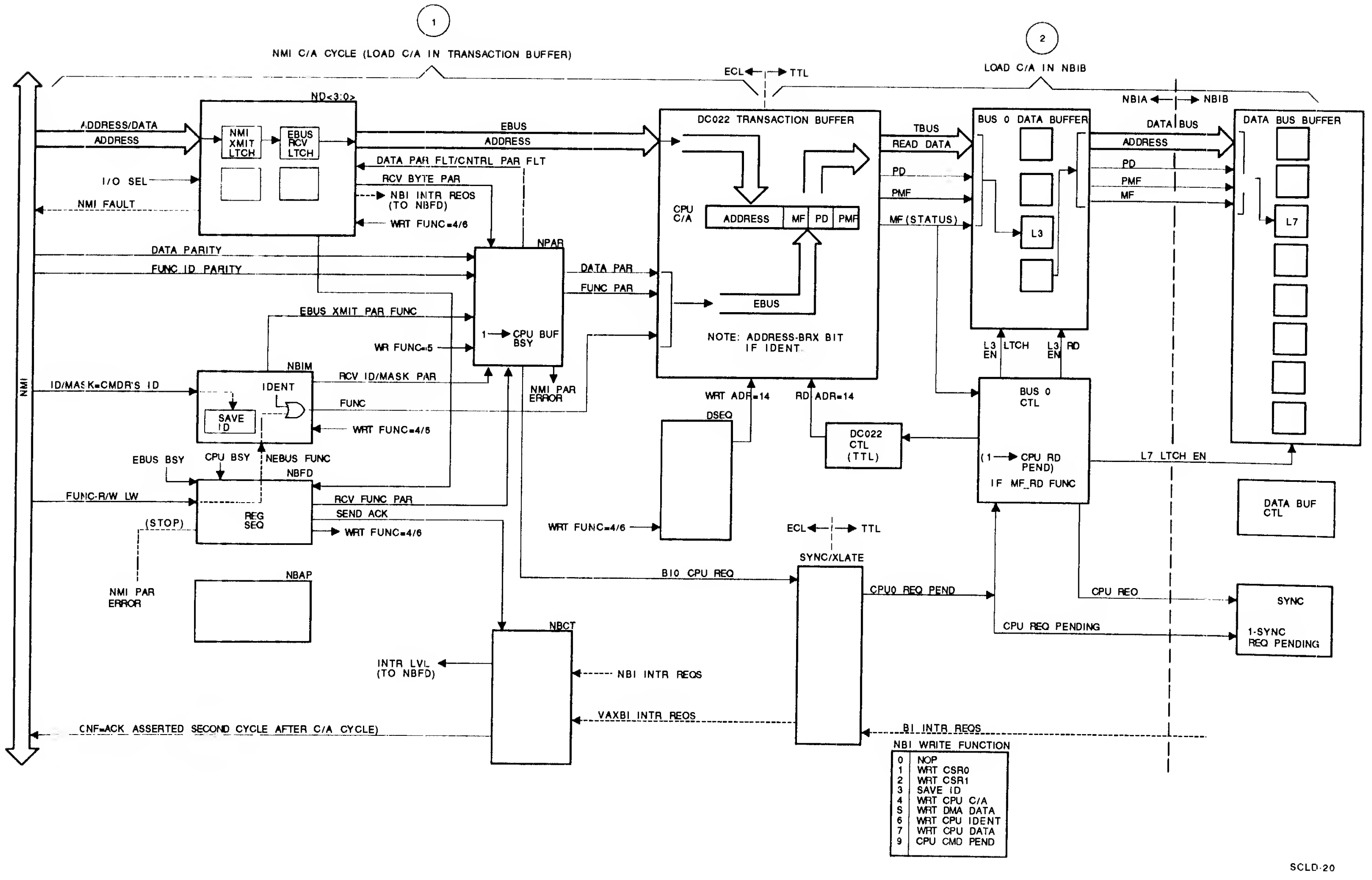
4. The NBIB's BIIC reads the command/address from the BCI data buffer, arbitrates for the VAXBI, and starts the transaction transmitting the command/address on the VAXBI.

Command/Address to Transaction Buffer

NBIA operation during the NMI command/address cycle is as follows:

1. Similar to a local read/write, the receive sequencer in the NBFD MCA advances state asserting SEND ACK and beginning the operation, provided the NMI address is selecting the NBIA, the NMI function is a read/write longword command, and there is no data or control line parity error. However, unlike the local read/write, CPU BUF BUSY and EBUS BUSY must both be deserted. If not, a busy response is returned to the CPU (SEND BUSY asserted). CPU BUF BSY indicates a VAXBI read/write or IDENT operation is already in progress. EBUS BUSY indicates that the NBIA has requested the NMI and is about to transfer a DMA command/address (and write data if a DMA write) to memory. EBUS BUSY also indicates that the NBIA has requested the NMI and is about to return read data to the CPU. (CSR BUSY is not checked, which means a VAXBI read/write or IDENT may be started while a local read is in progress.)

2. The NBFD MCA also gates the received NMI function to the NBIM MCA over the five NEBUS lines (it always does this) and asserts one of two NBI write functions, WRT CPU C/A (for VAXBI reads and writes) or WRT CPU IDENT.



0	NOP
1	WRT CSR0
2	WRT CSR1
3	SAVE ID
4	WRT CPU C/A
5	WRT DMA DATA
6	WRT CPU IDENT
7	WRT CPU DATA
9	CPU CMD PEND

Figure 3-11 NMI to VAXBI Command/Address Transfer (Sheet 1 of 2)

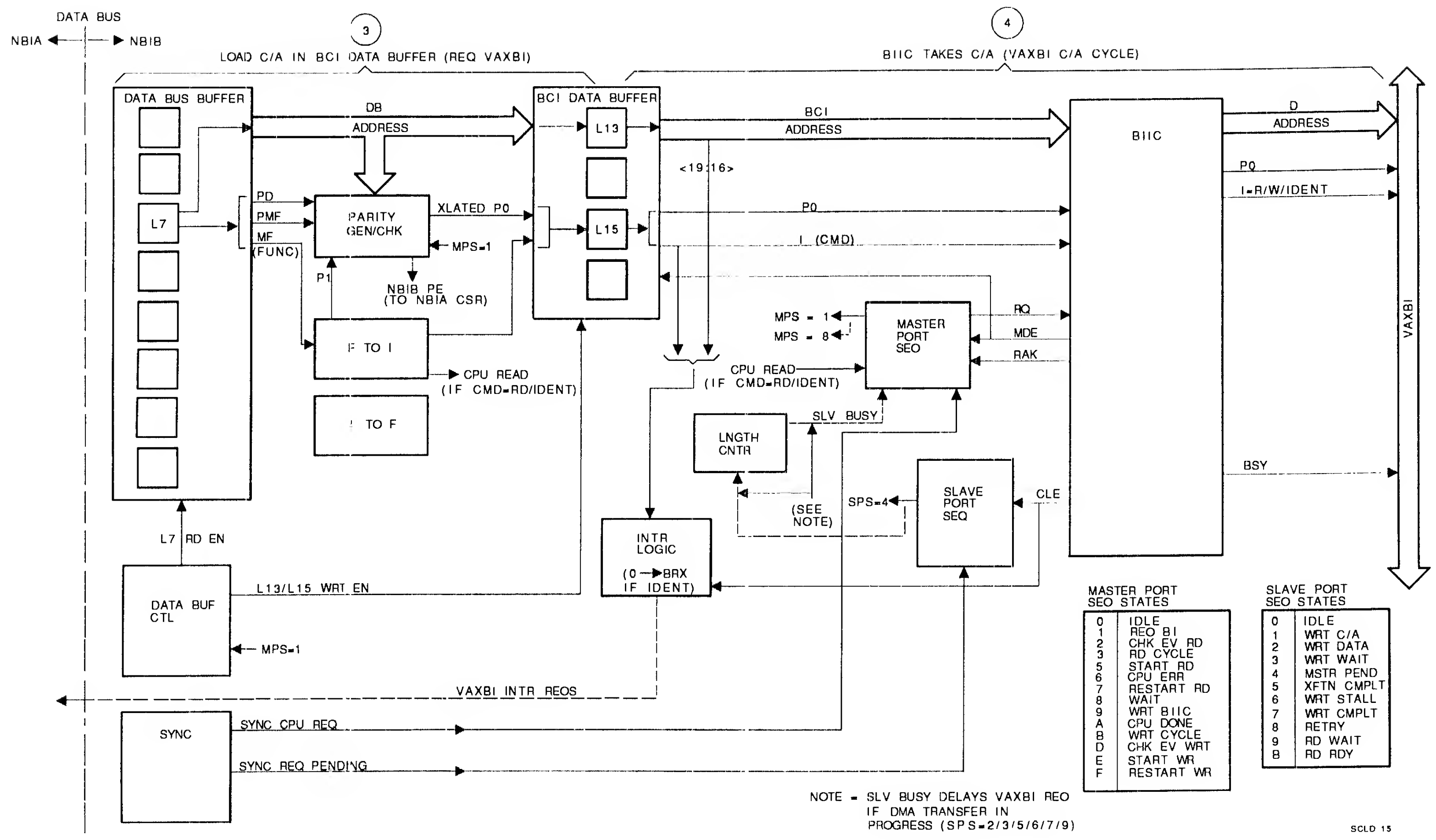


Figure 3-11 NMI to VAXBI Command/Address Transfer (Sheet 2 of 2)

The NBI write function then does the following:

1. In the NMI data buffer (ND MCAs), a WRT CPU C/A function causes the received NMI address (with bits <26:25> forced to zeros by ND3) to be transmitted on the EBus to the transaction buffer. A WRT CPU IDENT function causes just one bit to be transmitted to the transaction buffer (by ND2). This is either bit <19>, <18>, <17>, or <16> depending on the vector register (BRxVR) addressed. When later transmitted on the VAXBI as part of the IDENT command/address, this bit specifies the BR level being serviced.

2. In the NBIM MCA, a WRT CPU C/A function causes the received NMI function (gated from NBFD) to be transmitted on the EBus FUNC lines to the transaction buffer. A WRT CPU IDENT function, however, causes a function equal to 19 (hex) to be transmitted. The four low-order bits are later transmitted on the VAXBI function lines during the command/address cycle. (The IDENT command code is equal to 9.)

3. In the DSEQ MCA, either NBI write function sets CPU BUF BSY and causes the information from the ND and NBIM MCAs to be written into the transaction buffer location reserved for the CPU read/write command/address. Also, in the NPAR MCA, a CPU REQ signal is asserted. This request line (there is one for each of the VAXBIs that can be selected) signals the appropriate data bus controller (either 0 or 1) that a CPU command/address is in the transaction buffer and that it may be read and transferred to the associated NBIB.

Command/Address to NBIB

The CPU REQ signal, asserted as CPU REQ PEND after ECL to TTL translation and synchronization to the TTL clock, causes the data bus control to transfer the CPU command/address to the NBIB. However, the transfer will be delayed if the control is busy doing a DMA transfer. If (or when) the bus control is not busy, it does the following:

1. Reads the command/address from the transaction buffer and (by asserting a latch enable signal) latches it into the data bus buffer's first set of data latches (L1).

2. After loading L1, immediately asserts L1's read (transmit) enable and, on the data bus, the latch enable for L7 in the NBIB's data bus buffer. The command/address is then transferred over the data bus to the NBIB (into L7), independent of any NBIB control.

3. Asserts CPU REQ on the data bus to signal the NBIB that the CPU command/address has been loaded. Also sets an internal control bit (CPU RD PEND) if the command is a read function. (When a CPU read is pending and before the return read data is loaded in the NBIB, the bus control is free to handle DMA transfers. These can occur if the VAXBI read transaction initiated in the NBIB by the command/address has to be retried.)

In addition to activating the NBIA's data bus control as just described, the CPU REQ PEND signal is asserted on the data bus to prevent the NBIB's slave port sequencer from starting a DMA transfer (or fielding interrupts) until the end of the CPU read/write operation (in the NBIB).

Command/Address to BCI Data Buffer (Request VAXBI)

CPU REQ from the NBIA first causes the NBIB's master port sequencer to request a VAXBI transaction by the BIIC. The request will be delayed if the NBIB is busy doing a DMA operation (SLV BUSY = 1). (The SLV BUSY logic, located in the length counter, determines when a DMA operation is in progress by monitoring the slave port sequencer's present state.)

To make the VAXBI transaction request, the master port sequencer advances state (to the REQUEST BI state) asserting a code of 01 on the BCI RQ lines. The REQUEST BI state then causes the data buffer control to assert the read enable for L7 (in the data bus buffer) and also the write (latch) enable for L3 and L5 in the BCI data buffer. This transfers the command/address into the BCI data buffer in preparation for the upcoming VAXBI transaction.

When the command/address is passed between the data bus buffer and BCI data buffer, the address is loaded directly in L3 with no translation required. (The address is the VAXBI read/write address with a longword length code, or the IDENT BR level bit.) However, the command in the data bus buffer, which is the NMI or modified IDENT function code, is gated through the F to I logic before it is stored in L5. This is to translate the function code to the appropriate VAXBI command code.

Command/Address to BIIC (VAXBI Command/Address Cycle)

The outputs of L3 and L5 in the BCI data buffer connect directly to the BIIC. After the VAXBI transaction is requested by the master port sequencer, the BIIC asserts MDE causing the command/address now in L3 and L5 to be transmitted on the BCI data and command (I) lines. The BIIC then latches the command/address, arbitrates for the VAXBI, and begins the VAXBI read/write (or IDENT) transaction. During the first bus cycle, the command/address cycle, the BIIC asserts both RAK on the BCI and BSY on the VAXBI (both signals indicating a transaction is in progress) and transmits the command/address that it read from the BCI data buffer directly (unmodified) onto the VAXBI's data and I lines. (Note: A VAXBI address may be for one of the NBIB BIIC's own registers, in which case it will respond to the command/address cycle like any other VAXBI node. Furthermore, the NBIB BIIC may have an interrupt request pending and could respond to an IDENT.)

During any VAXBI command/address cycle, the BIIC asserts CLE on the BCI. In this case, when the command/address cycle is transmitted by the BIIC and the result of a CPU read/write operation (CPU REQ PEND = 1), the CLE signal causes the slave port sequencer to advance to the MASTER PENDING state. The sequencer then causes the BIIC to send RETRY responses to any DMA request (VAXBI read/write transaction) by a VAXBI device until the CPU read/write operation ends in the NBIB.

If the VAXBI transaction just initiated is an IDENT, CLE also causes the interrupt request being serviced (latched in the interrupt logic) to be cleared. The interrupt logic always latches and decodes the VAXBI command on the BCI I lines, and it also checks the state of BCI data lines <19:16>. It does this to identify interrupt requests (by incoming INTR and IPINTR transactions) from the VAXBI devices. In this case, when the NBIB has initiated the transaction and it is an IDENT, the asserted data line is used to clear the appropriate BR level.

3.3.3.2 Write Data Transfer -- Like the transfer of the command/address, the transfer of write data from the NMI to the VAXBI (Figure 3-12) consists of four basic operations.

1. The NBIA loads the NMI write data into its DC022 transaction buffer.

2. The NBIA reads the write data from its transaction buffer and loads it into the NBIB's data bus buffer (over the interconnecting data bus).

3. The NBIB reads the write data from its data bus buffer and loads it into its BCI data buffer.

4. The NBIB's BIIC reads the write data from the BCI data buffer and transmits it on the VAXBI (VAXBI write transaction in progress).

The end of the VAXBI write transaction ends the CPU write operation in the NBIB (and NBIA) unless a retry is requested by the addressed VAXBI node. Then, the NBIB simply requests another VAXBI transaction by the BIIC. The command/address and write data are buffered in the BIIC and do not have to be reloaded.

Write Data to Transaction Buffer

The loading of the NMI write data in the transaction buffer is very much like loading the command/address.

1. The received NMI function (this time, a write data function) advances the NBFD's receive sequencer causing the sequencer to assert an NBI write function.

2. The NBI write function (this time, WRT CPU DATA) then causes the NMI data buffer to transmit the received NMI write data (instead of a command/address) to the transaction buffer.

3. The NBI write function also causes the NBIM MCA to transmit the received NMI write mask (not the received NMI, or IDENT, function) to the transaction buffer.

4. Finally, and again like writing the command/address, the NBI write function causes the DSEQ MCA to load the write data and mask into a transaction buffer location. In this case, the location is the one reserved for both CPU write and read data.

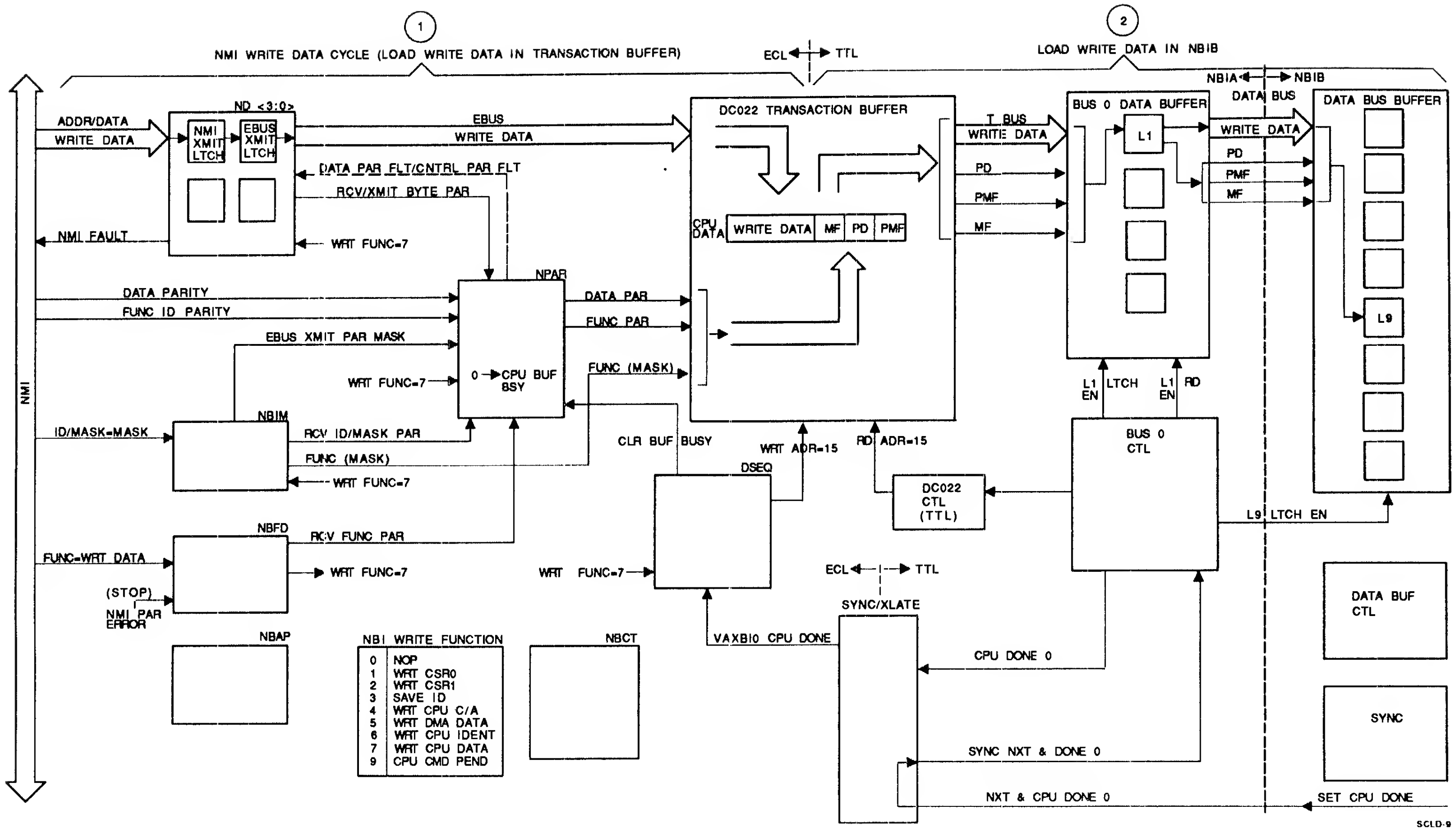
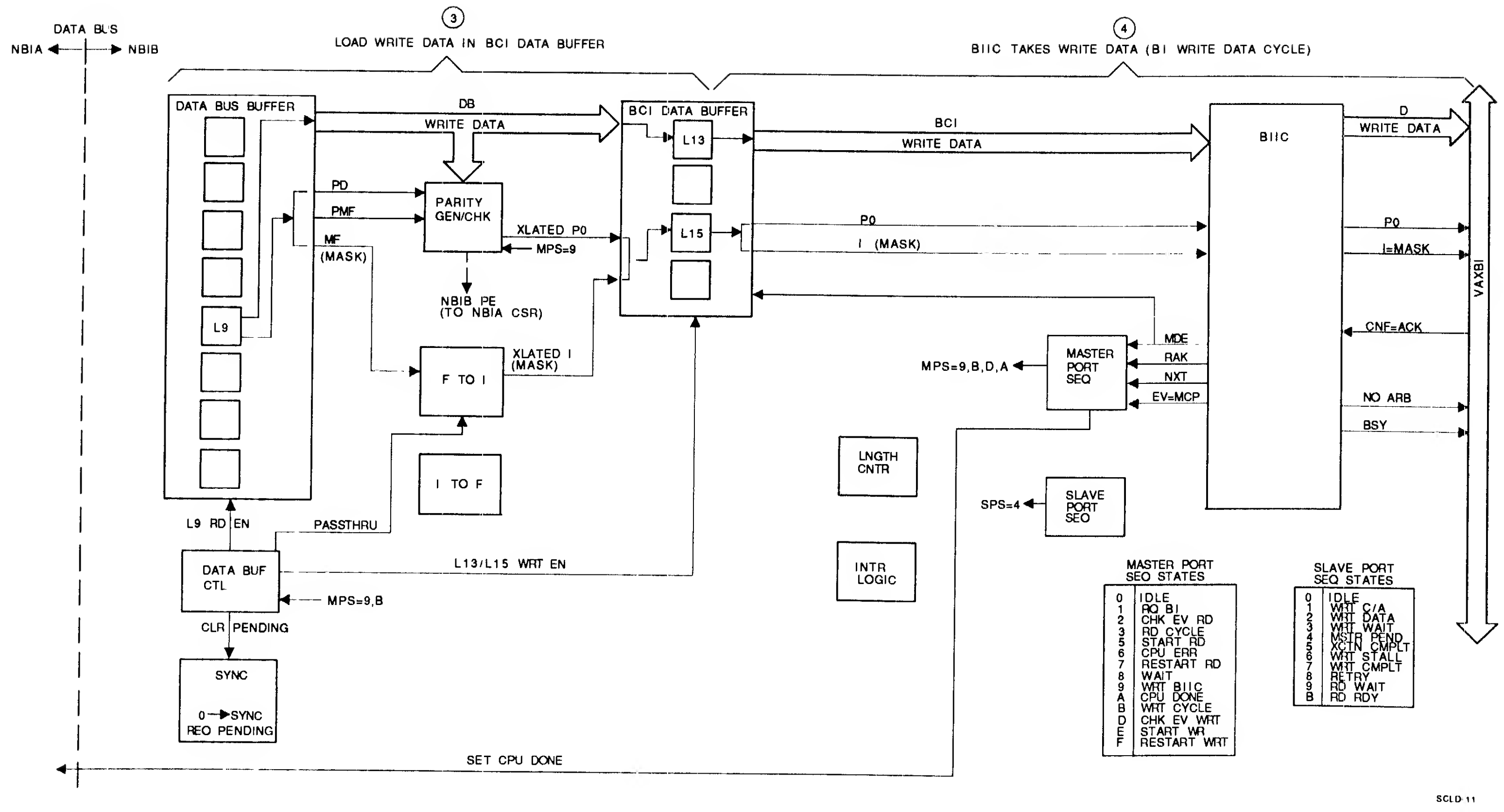


Figure 3-12 NMI to VAXBI Write Data Transfer (Sheet 1 of 2)



0	IDLE
1	RQ BI
2	CHK EV RD
3	RD CYCLE
5	START RD
6	CPU ERR
7	RESTART RD
8	WAIT
9	WRT BIIC
A	CPU DONE
B	WRT CYCLE
D	CHK EV WRT
E	START WR
F	RESTART WRT

0	IDLE
1	WRT C/A
2	WRT DATA
3	WRT WAIT
4	MSTR PEND
5	XCTN CMPLT
6	WRT STALL
7	WRT CMPLT
8	RETRY
9	RD WAIT
B	RD RDY

Figure 3-12 NMI to VAXBI Write Data Transfer (Sheet 2 of 2)

Write data to NBIB

There is one major difference between the transfer of write data to the NBIB and the transfer of the command/address to the NBIB. Although the data bus control (by reading the transaction buffer and by asserting the appropriate latch enables and read enable) still transfers the information to the NBIA's bus data buffer and then quickly to the NBIB's data bus buffer, it does not initiate the transfer in response to a request signal. (The command/address transfer was initiated by CPU REQ PENDING.) Instead, the transfer is initiated automatically by the bus control immediately following the transfer of the command/address. (Both the command/address and write data/mask are in the transaction buffer by the time CPU REQ PENDING is asserted.)

Another difference is that when the write data is loaded in the NBIB's data bus buffer, it is loaded in another set of latches (L9). (The command/address was loaded in L7.) Thus, the NBIA does not have to wait for the NBIB to transfer the command/address from L7 before it can transfer the write data. (The NBIB might be busy doing a DMA transfer.) Transferring both command/address and write data to the NBIB ends the CPU write data transfer in the NBIA. However, the NBIA's CPU BUF BSY signal must still be cleared (by CPU DONE) when the CPU write data transfer is completed by the NBIB. (The end of the VAXBI write transaction is discussed presently.)

Write Data to BCI Data Buffer

When the BIIC reads the command/address from the BCI data buffer and begins the VAXBI command/address cycle as previously described, the MDE signal (which gated the command/address to the BIIC) causes the master port sequencer to advance state after a delay. (A buffered MDE signal is used to advance the sequencer.) This sequencer state (WRITE BIIC) then causes the write data and mask now in the NBIB's data bus buffer (L9) to be transferred to the BCI data buffer overwriting the command/address (in L3 and L5.)

make the transfer, the NBIB's data buffer control asserts the necessary read and write (latch) enables as when transferring the command/address. Also, it asserts a PASSTHRU signal to the F to I logic because the write mask does not need to be translated for transmission on the VAXBI like the NMI (or modified IDENT) function in the command/address.

Write Data to BIIC (VAXBI Write Data Cycle)

When the BIIC is ready to take the write data and mask now in the BCI data buffer, it asserts NXT and MDE on the BCI. The NXT signal advances the master port sequencer (to the WRITE CYCLE state), and MDE gates the write and mask to the BIIC as it did the command/address. The BIIC then latches the write data and mask transmitting them on the VAXBI during the single write data cycle. The sequencer remains in the WRITE CYCLE state until the end of the VAXBI write transaction. In the normal case, this is when the addressed node has taken the write data and acknowledged that the transaction has completed successfully. It indicates this by returning an ACK response on the VAXBI CNF lines during the two bus cycles following the write data cycle. (The responding node may be the NBIB's own BIIC if it has been addressed.) If an addressed node is not ready to accept write data, it may return a STALL code on the CNF lines for one or more bus cycles before it finally takes the data and acknowledges that the transaction completed successfully. A transaction is also ended when the addressed node cannot accept write data and asserts a RETRY response on the CNF lines.

End of VAXBI Write Transaction (and Retries)

When the transaction ends, the BIIC deasserts RAK on the BCI. (RAK was asserted at the beginning of the command/address cycle). This causes the master port sequencer to check the event code asserted by the BIIC on the BCI EV lines. The sequencer also advances to the CHECK EVENT WRITE state. If the event code indicates the master port transaction completed successfully (MCP code), the sequencer advances state (to CPU DONE) asserting SET CPU DONE to the NBIA.

In the NBIA, and after synchronization to the NBIA's TTL clock, SET CPU DONE causes the bus control (that sent the command/address plus the write data and mask to the NBIB) to assert the CPU DONE signal. CPU DONE (after TTL to ECL translation and synchronization to the ECL clock) then causes CLR BUF BUSY to be asserted by the DSEQ MCA. This clears CPU BUF BSY in the NPAR MCA ending the CPU write operation and allowing the NBIA to acknowledge and execute the next CPU write (or read) to a VAXBI address.

When a VAXBI transaction ends and the event code (an RTR code) indicates a retry response was received, the master port sequencer immediately requests another VAXBI transaction by the BIIC. The command/address and write data/mask information does not have to be reloaded in the BIIC. The sequencer then returns to its WRITE CYCLE state when the transaction is retried on the VAXBI (BIIC asserts RAK again). As before, the sequencer remains in this state until the transaction ends. Also, as before, the sequencer then checks the event code and requests another VAXBI transaction if there is a retry response. This process is repeated for all retry responses that follow. When the addressed node finally takes the write data and acknowledges the transaction completed successfully, SET CPU DONE is asserted to end the CPU write operation.

The master port sequencer's WRITE CYCLE state causes SYNC REQ PENDING to be cleared, which returns the slave port sequencer to its IDLE state. The slave port sequencer can then respond to DMA requests with DMA transfers occurring during the time a VAXBI write transaction is being retried and before the CPU write operation ends.

3.3.3.3 Return Read Data Transfer -- The transfer of return read data (Figure 3-13) occurs at the end of the VAXBI read (or IDENT) transaction initiated by the command/address. (A VAXBI read transaction, but not an IDENT, may be retried before the transfer of return read data can start.) The return read data for an IDENT transaction is an interrupt vector.

Like the command/address and write data transfers, the transfer of return read data consists of four basic operations. However, data is passed in the opposite direction (VAXBI to NMI).

1. The NBIB loads the VAXBI return read data (received by its BIIC) into its BCI data buffer.

2. The NBIB transfers the return read data from the BCI data buffer into its data bus buffer.

3. The NBIA loads the return read data held in the NBIB's data bus buffer into its own transaction buffer.

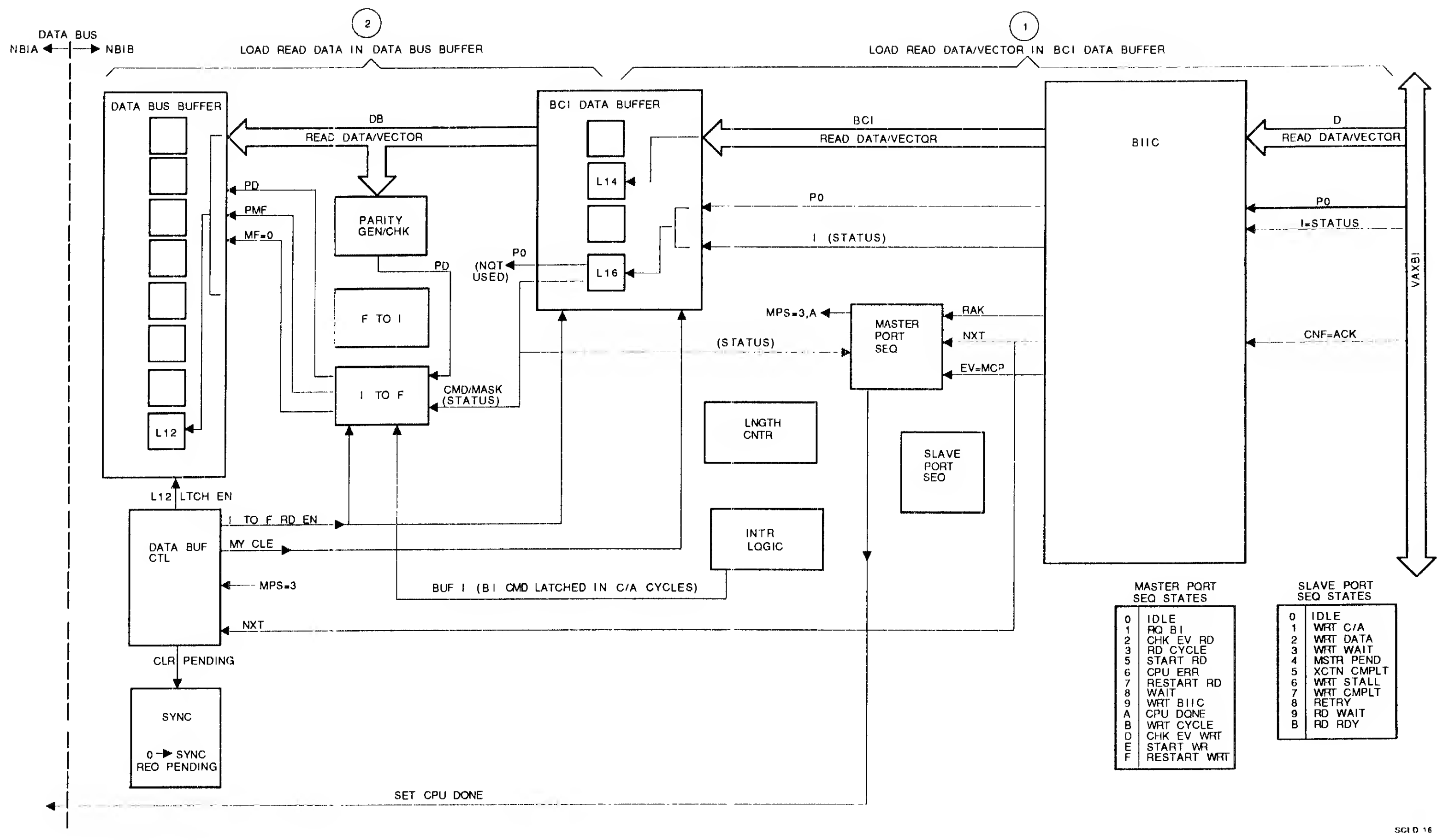
4. The NBIA requests the NMI, reads the return read data from the transaction buffer into its NMI data buffer (ND MCAs), and then transmits the data on the NMI when it wins the bus.

Read Data/Vector to BCI Data Buffer

Following the VAXBI command/address cycle (MDE and RAK asserted by the BIIC), the master port sequencer advances to the READ CYCLE state. This causes the NBIB's data buffer control to assert MY CLE, which, in turn, asserts the BCI data buffer's internal L4 and L6 latch enable. The latch enable is asserted continuously as long as MY CLE = 1, and even though no return read data has yet been received from the VAXBI.

When the BIIC receives the return read data from the responding VAXBI node during the VAXBI read data cycle (or IDENT vector cycle), it first transmits the data on the BCI and then asserts NXT (also on the BCI). (The NBIB's own BIIC may be the responding node.) Timing is such that the return read data is loaded in the BCI data buffer (in L4) before NXT is asserted. The status of the return read data, asserted on the VAXBI's I lines, is also received and loaded in the BCI data buffer (in L6).

A VAXBI return read data cycle, like a VAXBI write data cycle, can be delayed for one or more bus cycles. That is, when an addressed node is not ready to return the read data right away, it can send a STALL response on the CNF lines. Then, when ready, it sends the data together with an ACK response. (An IDENT vector cycle cannot be stalled.)



MASTER PORT SEQ STATES

0	IDLE
1	RQ BI
2	CHK EV RD
3	RD CYCLE
5	START RD
6	CPU ERR
7	RESTART RD
8	WAIT
9	WRT BIIC
A	CPU DONE
B	WRT CYCLE
D	CHK EV WRT
E	START WR
F	RESTART WRT

SLAVE PORT SEQ STATES

0	IDLE
1	WRT C/A
2	WRT DATA
3	WRT WAIT
4	MSTR PEND
5	XCTN CMPLT
6	WRT STALL
7	WRT CMPLT
8	RETRY
9	RD WAIT
B	RD RDY

Figure 3-13 VAXBI to NMI Return Read Data Transfer (Sheet 1 of 2)

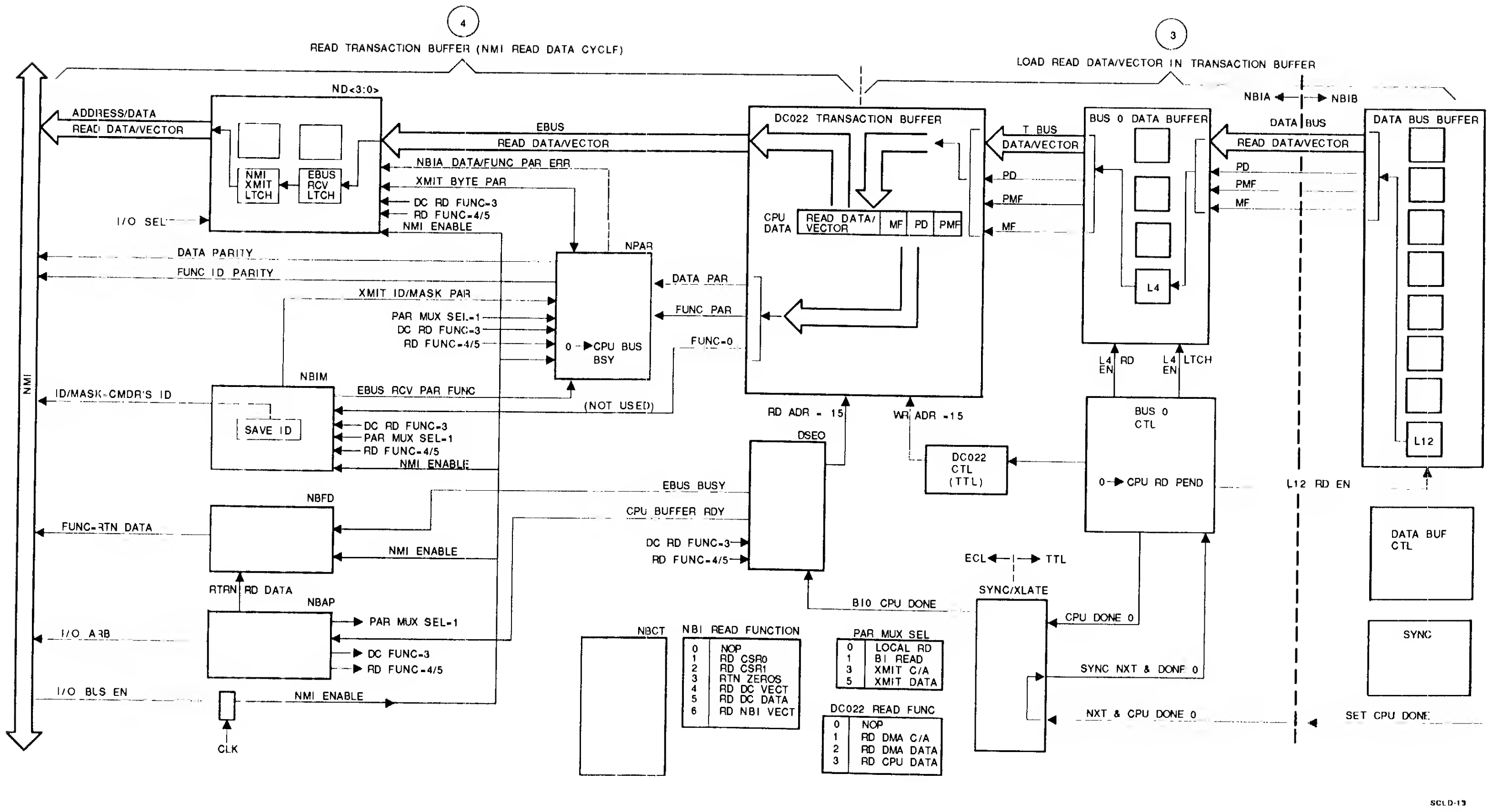


Figure 3-13 VAXBI to NMI Return Read Data Transfer (Sheet 2 of 2)

Read Data/Vector to Data Bus Buffer

Immediately after the BCI data buffer is loaded, the NXT signal causes the data buffer control to assert I TO F RD EN and a latch enable, transferring the return read data to the data bus buffer (to L2). The data status code, which is not used by the NBIA when it takes the data from L2, is not transferred. That is, the code is not passed through the NBIB's I to F logic. Zeros are loaded in L2, instead.

End of VAXBI Transaction (and Retries)

The BIIC acknowledges that it has taken the return read data from the VAXBI node by generating an ACK response on the CNF lines for the next two bus cycles. The BIIC also deasserts RAK (on the BCI) to indicate the end of the transaction. This causes the master port sequencer to check the event code asserted by the BIIC. If the event code is an MCP code, indicating the transaction completed successfully and good data was received, the master port sequencer asserts CPU DONE to the NBIA ending the CPU read operation in the NBIB.

A VAXBI read transaction, like a VAXBI write transaction, can be retried. (An IDENT transaction cannot be retried.) That is, a RETRY response on the VAXBI's CNF lines by the addressed node ends the current transaction. Then, when RAK is deserted and the master port sequencer checks the BIIC event code (an RCR code), the sequencer requests another VAXBI transaction and eventually returns to the READ CYCLE state when the transaction is retried (RAK = 1 again). The sequencer repeats this process for any additional retry responses. When the VAXBI node finally returns good read data, it is loaded in the NBIB and CPU DONE is asserted as previously described.

The master port sequencer's READ CYCLE state causes SYNC REQ PENDING to be cleared, allowing the slave port sequencer to advance to the IDLE state. The slave port sequencer can then respond to DMA requests with DMA transfers occurring when a VAXBI read transaction is being retried and before the CPU read operation ends in the NBIB.

Read Data/Vector to NBIA's Transaction Buffer

When the NBIB loads the return read data in its data bus buffer and asserts SET CPU DONE, the NBIA's data bus control (which sent the command/address to the NBIB) is normally in a wait state already generating the necessary read and latch enables to take the data. It may also be in an idle state (after transferring DMA data), in which case, the SET CPU DONE signal (after being synchronized to the NBIA's TTL clock, and with internal control bit CPU RD PEND set) causes the read and latch enables to be generated. In either case, the return data is transferred from the NBIB's data buffer (from L2) across the data bus to the NBIA's data buffer (to L4). Also, the synchronized SET CPU DONE signal causes the data bus control to write the data in the NBIA's transaction buffer and assert CPU DONE. The return read data is written in the transaction buffer location reserved for both CPU read and write data.

Read Data/Vector to NMI (NMI Data Cycle)

The CPU DONE signal, after TTL to ECL translation and synchronization to the ECL clock, causes the DSEQ MCA to assert CPU BUFFER RDY and EBUS BUSY, and to read the return read data from the transaction buffer. This will be delayed if a DMA command/address (and write data if a DMA write operation) is being written into memory, or if DMA read data is being returned from memory.

The CPU BUFFER RDY signal is an input to the transmit sequencer in the NBAP MCA causing it to request the NMI for a return data bus cycle and to advance state. The EBUS BUSY signal is used to prevent the receive sequencer (in the NBFD MCA) from starting another CPU read/write that requires a VAXBI transaction until the end of the operation, that is, when the return read data has been transmitted on the NMI. (CPU BUF BSY does the same thing, but it is cleared before the end of the operation.) The receive sequencer may begin a local read/write, however.

When the transmit sequencer requests the NMI (I/O ARB is asserted on the bus), it also asserts a DC022 read function (READ CPU DATA). This function does the following:

1. Causes the return read data (from the transaction buffer) to be loaded in the NMI data buffer's EBus receive latches (in the ND MCAs). The MF bits associated with the data (cleared by the NBIB) are also loaded in the NBIM MCA but are not used.

2. Causes CPU BUF BSY to be cleared in the NPAR MCA.

The transmit sequencer then asserts an NBI read function (either READ DC022 DATA or READ DC022 VECTOR), a PAR MUX SEL code equal to 1 (BI READ), and RTRN RD DATA. As for a local read, these signals select the information to be returned on the NMI. When the bus is granted to the NBIA (NMI ENABLE = 1), the following occurs:

1. The NMI data buffer transmits the return read data (now clocked into its NMI transmit latches) on the NMI data lines. If the return read data is a vector, and it is from a nonoffsetable VAXBI device (bits <13:9> = 0), the NMI data buffer transmits the vector offset value (in CSR0) on data lines (13:10>. It also transmits a 0 or 1 on data line <9> depending on whether it is NBIA0 or NBIA1.

2. The NBIM MCA transmits the ID saved during the command/address cycle on the NMI ID/MASK lines.

3. The NBFD MCA transmits a RTN DATA function on the NMI FUNC lines.

This ends the CPU read operation.

3.3.3.4 Parity Generation and Checking

Command/Address and Write Data/Mask Parity

During the command/address and write data cycles of NMI transactions that require a VAXBI transaction, NMI data and control line parity is generated and checked (against the received parity bits) as described for local read/write operations (Section 3.3.2.4). The NMI transaction is not acknowledged if a parity error is detected during the command/address cycle. Also, if the command/address has good parity but the write data has bad parity, no data is written in a register. That is, a CPU REQ is never generated and, thus, a VAXBI transaction never takes place.

Both a data parity bit (PD) and a control (mask/function) parity bit (PMF) are loaded in the transaction buffer and passed to the NBIB over the data bus along with the command/address or write data/mask. These bits (both even parity bits) are generated by the NPAR MCA.

To generate PD (address parity) and PMF (function parity) during a command/address transfer, the NPAR MCA does the following:

1. PD - Gates the NMI data parity bit to the transaction buffer if data bits <26> and <25> are both the same value. (These data bits are cleared by the NMI data buffer before they are loaded in the transaction buffer, so parity is not affected if both are 0 or both are 1.) If the two data bits are not the same value, the NMI data parity bit is inverted before it is loaded in the transaction buffer.

2. PMF - If the function is not an IDENT, gates the NMI FUNC parity bit computed by the NBIM MCA to the transaction buffer. (The received NMI control parity bit cannot be used in any way because it is computed for both the NMI FUNC and ID/MASK lines.) For an IDENT function (11001 loaded in transaction buffer), the NPAR MCA does not use the computed function parity bit. It simply makes PMF = 1.

To generate PD (write data parity) and PMF (mask parity) during a write data transfer, the NPAR MCA does the following:

1. PD - Gates the NMI data parity bit to the transaction buffer.

2. PMF - Gates the NMI ID/MASK parity bit computed by the NBIM MCA to the transaction buffer. (Again, the NMI control parity bit cannot be used because it is computed for both the NMI FUNC and ID/MASK lines.)

Data and control line parity are not checked in the NBIB until the command/address or write data/mask are transferred from the data bus buffer to the BCI data buffer. If a parity error is detected, the NBIB's parity generator/checker asserts an error signal (NBIB PE). A parity error does not prevent the operation from continuing. (The VAXBI transaction is executed.) However, NBIB PE sets an error bit in the NBIA (in CSR0) causing an NBI interrupt request. An error may be forced during diagnostic operations by setting a control bit (FORCE NBIB PE) in CSR1.

The parity generator/checker also generates a single parity bit (P0) for the command/address and write data/mask. This bit, the VAXBI odd parity bit, is loaded in the BCI data buffer and (along with the command/address or write data/mask) is passed to the BIIC and transmitted on the VAXBI during the VAXBI transaction. (The NBIB's BIIC is not enabled to generate the VAXBI parity bit for information supplied by the NBIA.)

P0 for write data/mask information is generated directly from the two parity bits sent by the NBIA (PD and PMF). However, the PMF bit cannot be used to generate P0 for a command/address. This is because the NMI function sent by the NBIA is translated to a VAXBI command as explained previously. As a result, the F to I logic regenerates the parity bit when it translates the NMI function. The parity generator/checker then uses this regenerated value (PI), together with PD, to generate parity for the command/address.

If the BIIC transmits bad parity during the VAXBI command/address cycle, it (and BIICs in other nodes connected to the VAXBI) set an internal bus error bit and generate an interrupt request by means of an INTR transaction. If the BIIC transmits bad parity during the write data cycle, it (and the BIIC in the node taking the data) set an internal error bit and generate an interrupt request. (The NBIB's BIIC could also be taking the data.)

RETURN READ DATA PARITY

Return read data parity is checked by the BIIC when the data is first taken from the VAXBI. It is not checked again until the return read data is read from the transaction buffer in the NBIA. If the BIIC detects an error, it sets an internal error bit and generates an interrupt request.

To generate both data and control parity (PD and PMF) bits for transmission to the NBIA (the single P0 parity bit received from the VAXBI and asserted by the BIIC is not used), the NBIB does the following:

1. PD - The parity generator/checker computes even parity for the VAXBI data when the data is transferred to the NBIB's data bus buffer. The parity bit is gated to the data bus buffer (along with the PMF bit) by the NBIB's I to F logic.

2. PMF - When the I to F logic gates the PD bit to the data bus buffer, it gates a PMF (even parity) bit equal to zero. This is because no VAXBI control information (read status) is loaded in the data bus buffer. Zeros are loaded and taken by the NBIA along with the return read data as explained previously.

The NBIB generates bad (odd) parity for both the data and control information (zeros) taken by the NBIA if the FORCE NBIB PE bit in CSR1 is set.

In the NBIA, when the return read data is read from the transaction buffer, the following occurs:

1. The NMI data buffer generates parity for each data byte (the data is now in the buffer's EBus receive latches) and the NBIM generates an even parity bit for the control information (zeros in this case).

2. The NPAR MCA then completes the data parity generation and checks the result with the data parity bit (PD) read from the transaction buffer. Also, it compares the generated control parity bit with the control parity bit (PMF) read from the transaction buffer. If an error is detected, NBIA DATA PAR ERR or NBIA FUNC PAR ERR sets the corresponding parity error bit in ND1 (CSR0) or ND0 (CSR1), causing an NBI interrupt request to be asserted.

The NPAR MCA also generates the NMI data and control line parity transmitted during the NMI return data cycle. It uses PD from the transaction buffer to generate the data line parity. However, similar to a local read, it uses the parity bit generated by the NBIM MCA (which transmits the saved ID on the NMI) to generate control line parity.

3.3.4 Write Sequence Faults

A write sequence fault can be detected during a CPU write operation. If an NMI write data cycle does not immediately follow the NMI command/address cycle, the NBFD MCA's receive sequencer (which is waiting for the write data) asserts WRT SEQ FLT and advances to the idle state. This signal causes the corresponding error bit to be set in CSR0. The error bit, in turn, asserts the NBI's FAULT DETECT line on the NMI, which causes a CPU interrupt request to be generated. The write sequence fault ends the operation with no data being written in an NBIA register (during a local write) or the transaction buffer (for a VAXBI write). Also, for a VAXBI write, timing is such that WRT SEQ FLT prevents the NPAR MCA from asserting a CPU request signal and, thus, a VAXBI transaction is not initiated.

3.3.5 NMI Bus Access Timeouts

When the NBAP MCA requests the NMI in order to return read data, and the bus is not granted after two system slow clock periods (approximately six milliseconds at the normal system clock rate), the MCA's bus access timeout counter asserts an error signal. This signal (BUS ACCESS TMOUT) connects to the timeout encoder in the NBCT MCA. The encoder then asserts the appropriate timeout error code, which is loaded in CSR0 causing an NBI interrupt request to be asserted.

3.3.6 VAXBI Errors

Once the NBIB has initiated a VAXBI transaction on the VAXBI (RAK asserted by the BIIC), any BIIC event code other than an MCP (transaction completed successfully) or a RCR (retry) code causes the master port sequencer to assert SET CPU ERROR. (SET CPU DONE is not asserted.) Some of the events that can cause the CPU error condition are as follows:

1. Read data substitute (RDS) error - Addressed node returned bad (uncorrectable) read data/vector.

2. Bad parity received (BPR) - Returned read data/vector had bad parity.

3. Retry timeout (RTO) - Addressed node returned 4096 consecutive retry responses.

NOTE

Refer to the event code descriptions in Table 3-1 to see all the error conditions that can be detected by the BIIC.

Also, the master port sequencer asserts SET CPU ERROR if a VAXBI transaction has been requested but not yet initiated (RAK still deserted), and a bus timeout occurs. (The BIIC asserts a BTO event code.) The bus timeout condition is when the BIIC cannot get access to the VAXBI after 4096 bus cycles.

In all cases, when the BIIC's event code causes SET CPU ERROR to be asserted, the BIIC sets an internal error bit to identify the failure and generates an interrupt request.

SET CPU ERROR ends a CPU read/write operation as follows:

1. Causes the data bus control in the NBIA to assert CPU ERROR. The control also returns to idle state if the operation is a CPU read.

2. For a CPU write, CPU ERROR then ends the operation that occurs normally when CPU DONE is asserted. That is, it causes the DSEQ MCA to clear CPU BUF BSY (in the NPAR MCA) leaving the NBIA free to acknowledge and execute the next CPU read/write to a VAXBI address.

3. For a CPU read, CPU ERROR causes the DSEQ MCA to clear CPU BUF BSY and assert RTRN ZEROS. This last signal causes the transmit sequencer in the NBAP MCA to request the NMI. (CPU DONE sets CPU BUF RDY to cause a bus request during a normal CPU read.) The transmit sequencer then asserts an NBI read function (RTRN ZERO) causing the NMI data buffer to transmit zeros on the NMI once the bus is granted and NMI ENABLE asserted. No data is read from the transaction buffer as in the normal case. Except for the return of zeros, the NMI data cycle is a normal one with the commander's (saved) ID transmitted on the NMI ID/MASK lines and an RTN RD DATA function transmitted on the NMI FUNC lines.

In one special case, a CPU read/write operation is terminated by a CPU error condition not detected by the NBIB. Actually, it occurs when an addressed NBIB is not connected to the NBIA (data bus input BI PRESENT = 0). In this case, the NBIA's corresponding data bus controller asserts CPU ERROR shortly after reading the command/address from the transaction buffer. The operation then ends as previously described.

3.4 DMA READ/WRITE OPERATIONS

A VAXBI device can access the system's memory (on the NMI) through the NBI. The range of accessible memory addresses is specified by the values loaded (under program control) in the starting and ending address registers located in the NBIB's BIIC. All memory addresses, by definition, have address bit <29> equal to 0.

When a VAXBI read/write transaction makes a memory reference within the specified range of addresses, the NBIB responds by first taking the command/address (and the write data for a VAXBI write transaction). It then stalls the VAXBI transaction. The NBIA responds by first taking the command/address (and write data) from the NBIB, and then initiating an NMI read/write transaction. After the NBIA takes all the write data, the NBIB ends the VAXBI write transaction. This is before the data has been written into memory (a disconnected write). A VAXBI read transaction continues to be stalled.

If an NMI write transaction is initiated by the NBIA, it transmits the write data to memory immediately after sending the command/address. This ends the NMI write transaction and completes a DMA write operation.

If an NMI read transaction is initiated by the NBIA, the transaction ends when the memory returns the read data to the NBIA. The data is then passed to the NBIB, which transmits it on the VAXBI ending the (stalled) VAXBI read transaction. This completes a DMA read operation.

DMA read/write operations are summarized in Table 3-4. As shown, a VAXBI transaction (which may specify a longword, quadword, or octaword transfer), initiates an NMI transaction of the same length with one exception. The exception is for a VAXBI read quadword transaction. Because there is no NMI read quadword transaction, an NMI read octaword transaction is initiated instead. The NBIB then returns only the addressed quadword to end the VAXBI transaction. As shown in Figure 3-14, the addressed quadword will be the first two longwords in the octaword read from memory if the address is aligned. However, if the quadword address is unaligned (a wrapped quadword read), the quadword will be the first and last longwords read from memory.

Table 3-4 DMA Read/Write Summary

VAXBI Transaction (Note 1)	Data Length	NBI Response	Resulting NMI Transaction (Note 2)	See Note
WR/WCI	L	STALL --> [ACK,ACK,ACK]/RETRY	WL	3
	QW	ACK,STALL --> [ACK,ACK,ACK]/RETRY	WQM	3,4
	OW	ACK,ACK,ACK,STALL --> [ACK,ACK,ACK]/RETRY	WO	3
UWMCI	L	STALL --> [ACK,ACK,ACK]/RETRY	WLUM	3
	QW	ACK,STALL --> [ACK,ACK,ACK]/RETRY	WQUM	3
	OW	ACK,ACK,ACK,STALL --> [ACK,ACK,ACK]/RETRY	WOUM	3
WMCI	L	STALL --> [ACK,ACK,ACK]/RETRY	WLM	3
	QW	ACK,STALL --> [ACK,ACK,ACK]/RETRY	WQM	3
	OW	ACK,ACK,ACK,STALL --> [ACK,ACK,ACK]/RETRY	WOM	3
RD/RCI	L	STALL --> ACK/RETRY	RL	6
	QW (A)	STALL --> [ACK,ACK]/RETRY	RO	5,6
	QW (UA)	STALL --> [ACK,STALL,STALL,ACK]/RETRY	RO	5,6
	OW	STALL --> [ACK,ACK,ACK,ACK]/RETRY	RO	6
IRCI	L	STALL --> ACK/RETRY	RLI	6
	QW (A)	STALL --> [ACK,ACK]/RETRY	ROI	5,6
	QW (UA)	STALL --> [ACK,STALL,STALL,ACK]/RETRY	ROI	5,6
	OW	STALL --> [ACK,ACK,ACK,ACK]/RETRY	ROI	6

NOTES:

1. VAXBI Transaction types:

WR = Write	RD = Read
WCI = Write (Cache Intent)	RCI = Read (Cache Intent)
UWMCI = Unlock Write Mask (Cache Intent)	IRCI = Interlock Read (Cache Intent)
WMCI = Write Mask (Cache Intent)	

2. NMI Transaction Types:

   WL/WO = Write (Longword/Octaword)
   WLM/WQM/WOM = Write (Longword/Quadword/Octaword) Masked
   WLUM/WQUM/WOUM = Write (Longword/Quadword/Octaword) Unlock Masked
   RL/RO = Read (Longword/Octaword)
   RLI/ROI = Read (Longword/Octaword) Interlocked

3. NBIB stalls VAXBI write transaction until NBIA takes command/address and write data. NBIA retries NMI write transaction if no response from memory, or if MEMORY BUSY asserted during command/address or first data cycle.

4. Write mask set to all ones by NBIB hardware.

5. VAXBI read quadword transaction causes octaword to be read from NMI memory. (No NMI quadword read transaction implemented.) The quadword read may be aligned (A) or unaligned (UA).

6. NBIB stalls VAXBI read transaction until NBIA returns read data from memory. NBIB also stalls transaction between data cycles of an unaligned QW read due to the reordering of data. NBIA retries NMI read transaction if MEMORY BUSY asserted during command/address cycle. NBIB requests retry of VAXBI read transaction if bus access timeout or retry timeout in NBIA.

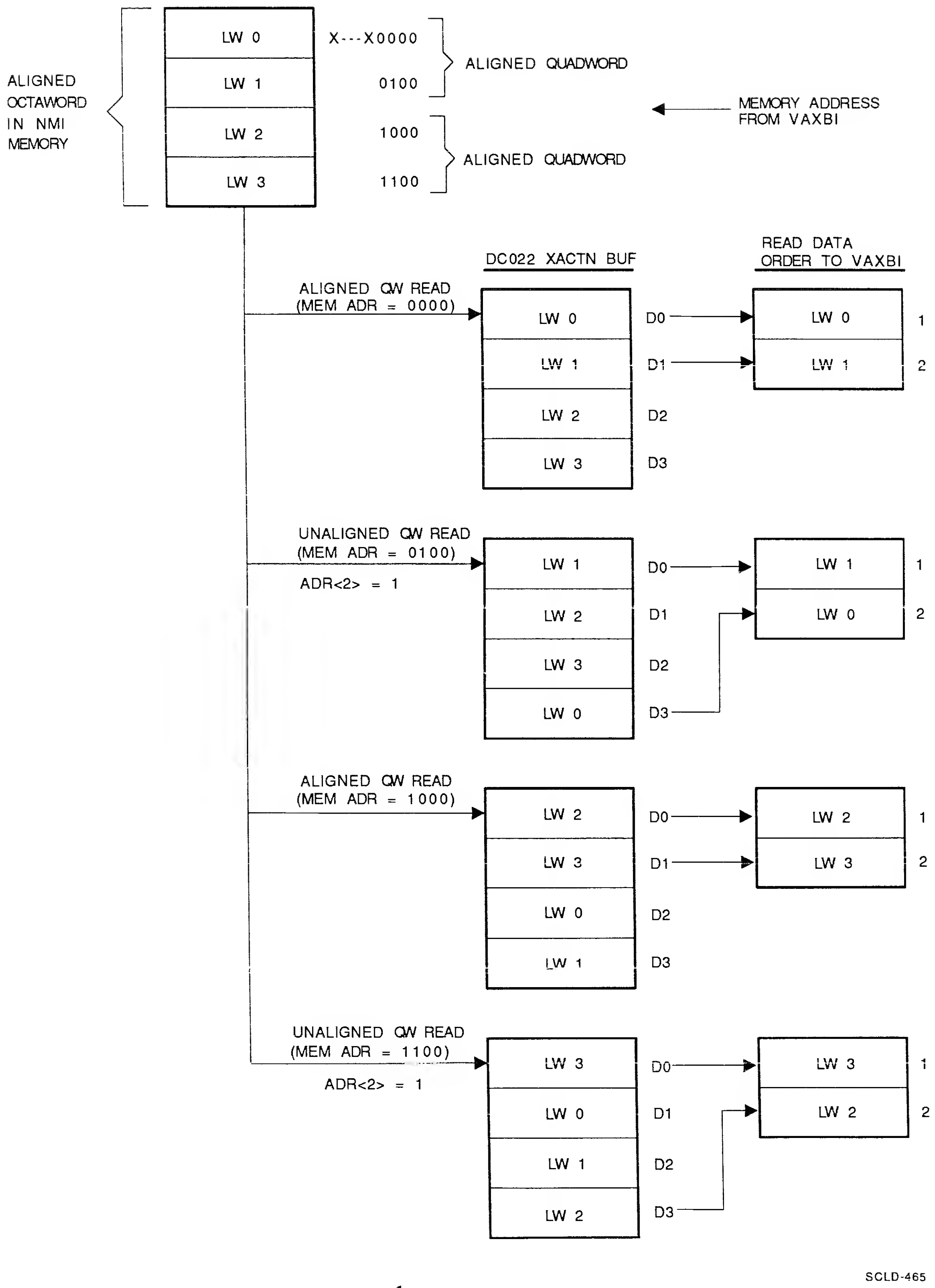

ALIGNED OCTAWORD IN NMI MEMORY
LW 0
LW 1
LW 2
LW 3
X---X0000
0100
1000
1100
ALIGNED QUADWORD
ALIGNED QUADWORD
MEMORY ADDRESS FROM VAXBI
DC022 XACTN BUF
READ DATA ORDER TO VAXBI
ALIGNED QW READ (MEM ADR = 0000)
UNALIGNED QW READ (MEM ADR = 0100)
ADR<2> = 1
ALIGNED QW READ (MEM ADR = 1000)
UNALIGNED QW READ (MEM ADR = 1100)
ADR<2> = 1
D0
D1
D2
D3
SCLD-465

Figure 3-14 Aligned and Unaligned Quadword Read Data Ordering

Referring again to Table 3-4, you can see that VAXBI read/write transactions with cache intent are treated as ordinary reads and writes. (The system does not support cached VAXBI operations.) Also, a write mask, interlock read, or unlock write mask VAXBI transaction initiates a corresponding NMI transaction (write masked, read interlocked, write unlock masked). However, an ordinary quadword write (VAXBI WR/WCI transaction) causes a masked quadword write on the NMI. (All NMI write quadword transactions are masked writes.) As a result, for these VAXBI transactions, the NBIB forces the mask bits (received with the write data from the VAXBI) to all ones before they are passed to the NBIA.

The transfer of the command/address and the data between the VAXBI and NMI during DMA read/write operations is shown in Figure 3-15.

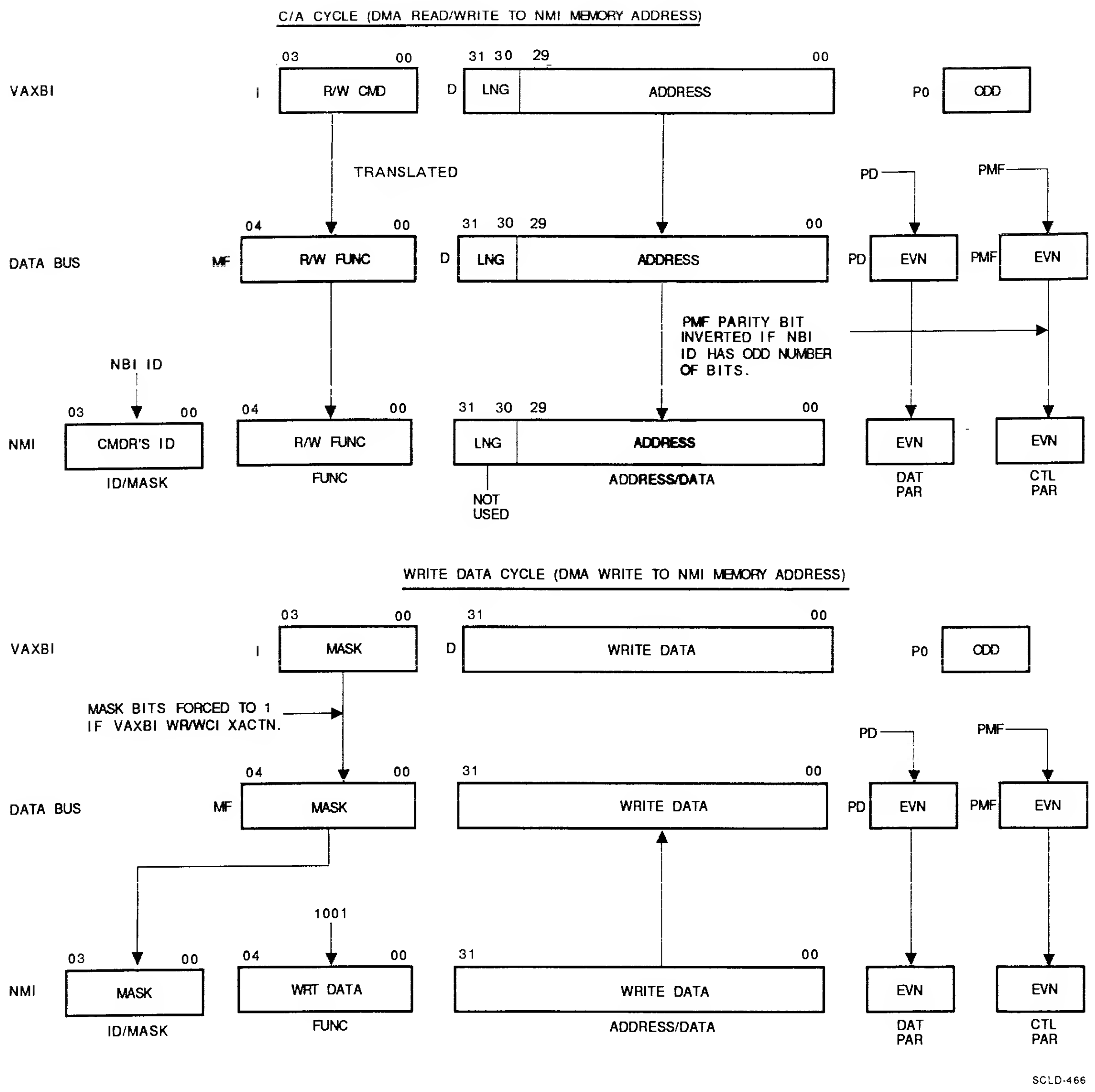


Figure 3-15 Basic Information Flow Between VAXBI and NMI During DMA Read/Write Operations (Sheet 1 of 2)

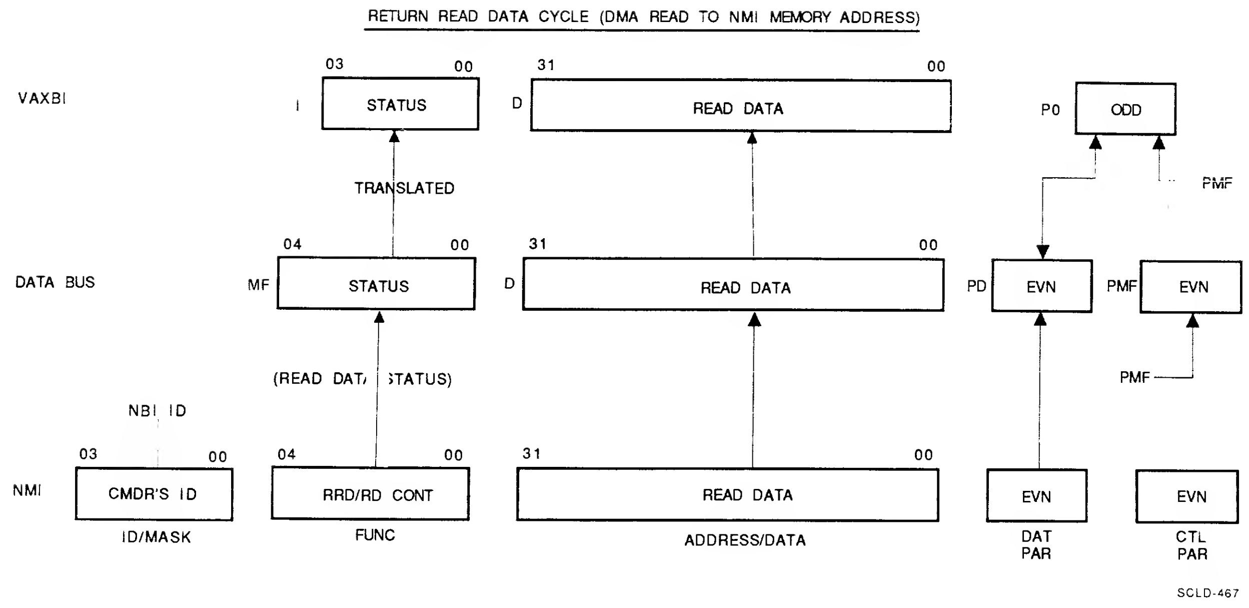


Figure 3-15 Basic Information Flow Between VAXBI and NMI During DMA Read/Write Operations (Sheet 2 of 2)

3.4.1 Command/Address Transfer

The transfer of the command/address from the VAXBI to the NMI (Figure 3-16) consists of three basic operations.

1. The NBIB transfers the VAXBI command/address received by its BIIC into its BCI data buffer and then immediately into its data bus buffer.

2. The NBIA transfers the command/address held in the NBIB's data bus buffer into its own transaction buffer.

3. The NBIA requests the NMI, reads the command/address from its transaction buffer into its NMI data buffer, and then transmits the command/address on the NMI when it wins the bus. During a DMA write operation, the requesting of the NMI and the reading of the command/address into the NBIA's transaction buffer does not occur until the NBIA has transferred all the write data from the NBIB. (Refer to Section 3.4.2.)

3.4.1.1 Command/Address to BCI Data Buffer and Data Bus Buffer -- When the BIIC detects a VAXBI command/address cycle, it asserts CLE and transmits the command/address on the BCI. The CLE signal does the following:

1. Causes the VAXBI command/address transmitted by the BIIC to be loaded in the BCI data buffer. The VAXBI address on the BCI data lines is loaded in one set of latches (L4). The VAXBI command on the BCI I lines is loaded in another (L6).

2. In addition, latches the VAXBI command on the I lines in a set of latches located in the interrupt logic. The latch outputs, (BUF I<3:0>), connect to decoding circuitry in the I to F logic.

3. Also stores the state of BCI data lines <31:30> and data line <2> in latches (BUF D <31:30> and <2>) located in the length counter logic.

CLE also advances the slave port sequencer to the WRITE C/A state unless the NBIB is about to do a VAXBI read/write transaction as the result of a CPU read/write request. In this case, the slave port sequencer will be locked in (or advance to) the MASTER PENDING state. The sequencer is locked in the MASTER PENDING state by CPU REQ PENDING as discussed in Section 3.3.3.1.

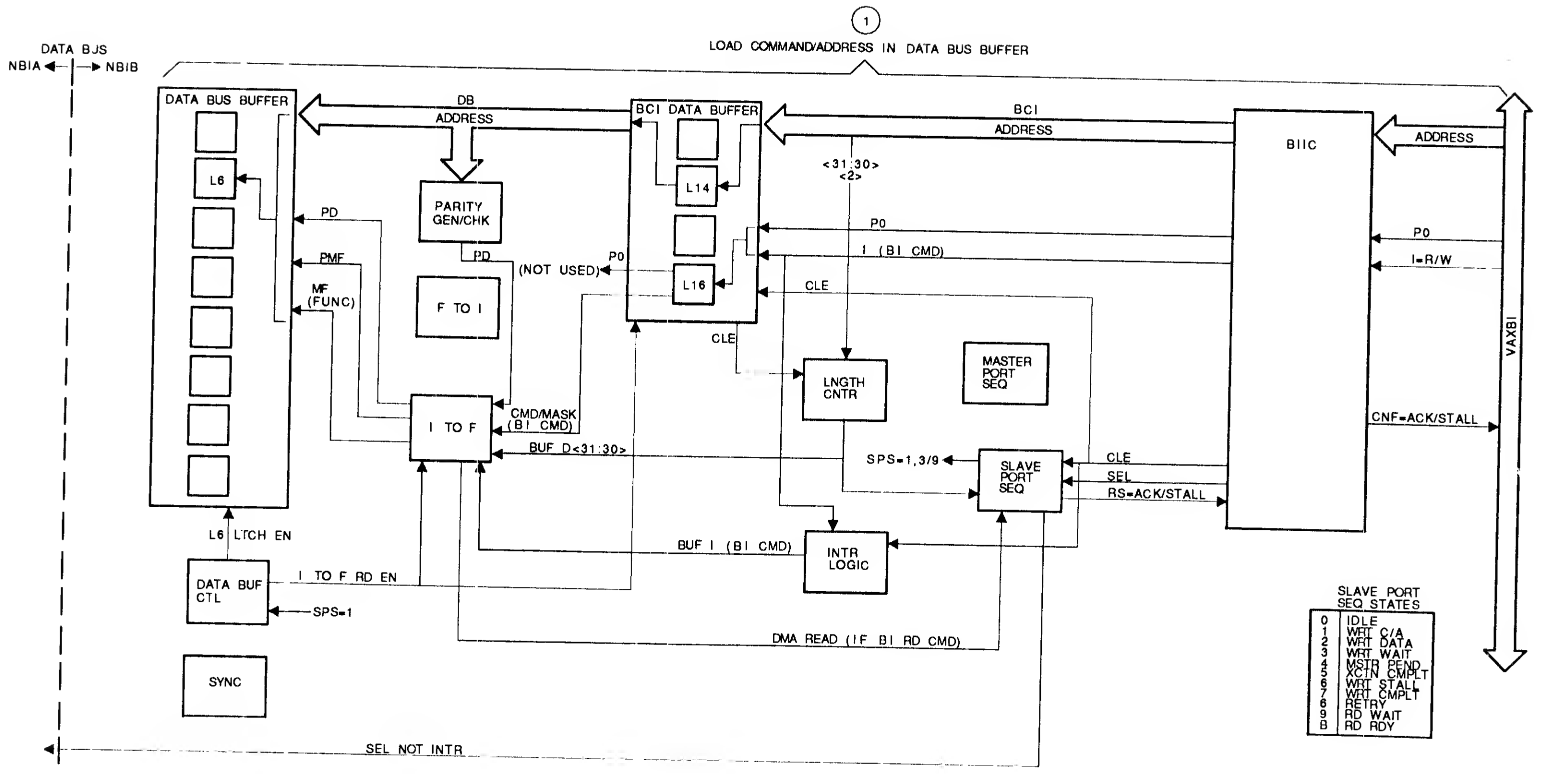


Figure 3-16 VAXBI to NMI Command/Address Transfer (Sheet 1 of 2)

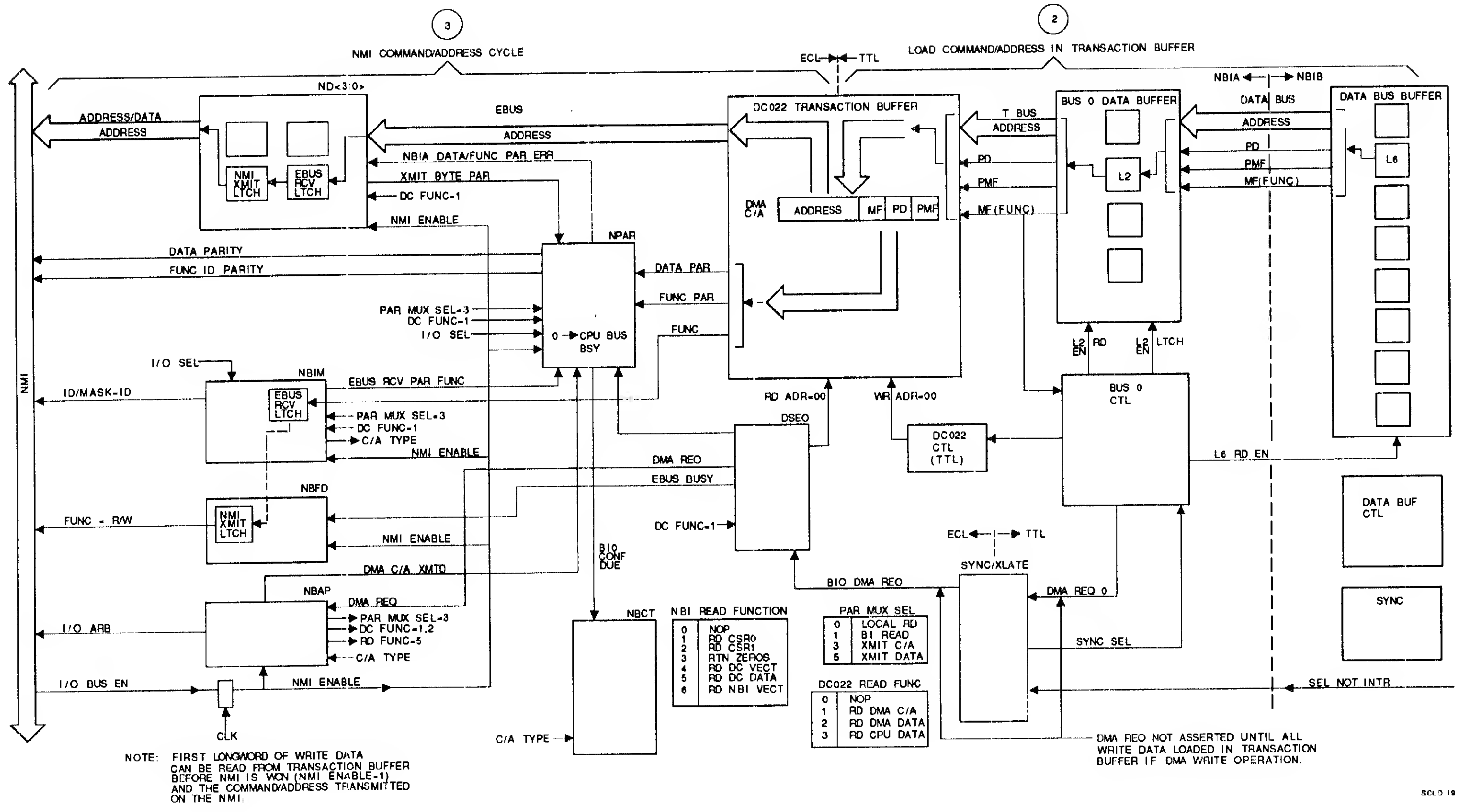


Figure 3-16 VAXBI to NMI Command/Address Transfer (Sheet 2 of 2)

When the sequencer advances to the WRITE C/A state, the data buffer control asserts I to F RD EN and a latch enable moving the command/address just loaded in the BCI data buffer into the data bus buffer (in L6). The VAXBI command is passed through the I to F logic and (if a read/write) is translated to the appropriate NMI command. The command/address is then ready to be taken by the NBIA provided it is a read/write directed to NMI memory. That is, the address is a memory address falling within the range specified by the BIIC's starting and ending address registers. If this is the case, the BIIC asserts SEL immediately after it asserts CLE.

SEL then advances the slave port sequencer to either the READ WAIT or WRITE WAIT STATE. The state depends on whether decoder output DMA READ is asserted or not. (This signal is from the circuitry located in the I to F logic that is decoding the just latched VAXBI command, BUF I <3:0>). The sequencer then does the following:

1. Asserts SEL NOT INTERRUPT (a DMA request) to the NBIA.

2. Transmits an ACK or STALL code on the RS lines to the BIIC. This determines the code transmitted by the BIIC on the VAXBI CNF lines during the first data cycle of the transaction. If a read, a STALL code is transmitted during the first cycle (and all others until return read data is returned by the NBIA and transmitted on the VAXBI). If a write, the code transmitted depends upon the data length of the transaction. If the length is one longword, a STALL code is also transmitted during the first cycle. If the length is a quadword or octaword, an ACK code is transmitted. (The data length, BUF D<31:30>, was previously latched in the data length counter logic.)

If the slave port sequencer is in the MASTER PENDING state when SEL is asserted by the BIIC, the slave port sequencer requests a retry of the DMA read/write transaction (asserts a RETRY code on the RS lines to the BIIC). Retry requests (a RETRY code on the CNF lines) will continue to be returned to the VAXBI device (if it retries the transaction) until CPU REQ PENDING is cleared. That is, when the slave port sequencer can move to the WRITE C/A state and accept the DMA read/write request. CPU REQ PENDING is cleared when the VAXBI transaction initiated by the CPU read/write request has either completed successfully or has, itself, received a retry request. In other words, a DMA read/write can be accepted by the NBIB when a CPU read/write is in the process of being retried. This prevents one device (by sending retry requests) from locking out another needing to make a DMA read/write.

3.4.1.2 Command/Address to NBIA's Transaction Buffer -- When the SEL NOT INTERRUPT signal is received from an NBIB (and synchronized to the NBIA's TTL clock), the associated data bus controller in the NBIA asserts the necessary read and latch enables to transfer the command/address from the NBIB's data bus buffer (from L6) to the NBIA's data buffer (to L2). Then, if internal control bit DMA BUF BSY is not set, it writes the command/address into the transaction buffer. There are two transaction buffer locations reserved for a DMA read/write command/address, one for each VAXBI (NBIB).

After the command/address is written in the transaction buffer, the data bus control (either 0 or 1) sets its DMA BUF BSY control bit and also asserts DMA REQ n (n = 0 or 1) if the operation is a DMA read. If the operation is a DMA write, BIn DMA REQ is not asserted until the write data is taken from the NBIB. (Refer to Section 3.4.2.)

The DMA REQ n signal is used to initiate the NMI transaction to read/write memory. The now set DMA BUF BSY control bit prevents the data bus control from writing another command/address in the transaction buffer, and asserting another DMA request, until memory responds to the NMI transaction initiated by the first. That is, until memory acknowledges a previous write transaction, or until it returns the read data requested by a previous read transaction.

3.4.1.3 Command/Address to NMI (NMI Command/Address Cycle) -- To initiate the NMI transaction, the DMA REQ n signal is first translated from TTL to ECL and synchronized to the ECL clock. It then causes the DSEQ MCA to assert DMA REQ and EBUS BUSY, and to read the DMA command/address from the transaction buffer. This will be delayed if the NBIA is already engaged in an NMI transfer due to another DMA request (initiated by the other NBIB) or a CPU read/write operation.

Once asserted, EBUS BUSY prevents the receive sequencer in the NBFD MCA from beginning any CPU read/write transaction requiring a VAXBI transfer (a local read/write may be started), and DMA REQ causes the transmit sequencer in the NBAP MCA to request the NMI (assert I/O ARB), assert a PAR MUX SEL code (TRANSMIT C/A), and assert a DC022 read function (READ DMA C/A). The read function then causes the following:

1. The address read from the transaction buffer is loaded in the NMI data buffer's EBus receive latches.

2. The translated VAXBI (to NMI) read/write function read from the transaction buffer is latched in the NBIM MCA.

The DSEQ MCA then reads the first DMA data location in the transaction buffer, and the transmit sequencer again asserts another DC022 read function (READ DMA DATA) plus an NBI read function (READ DC022 DATA) and a PAR MUX SEL code (TRANSMIT C/A). The following then occurs while the address that was just read from the transaction buffer is clocked into the NMI data buffer's NMI transmit latches (from its EBus receive latches), and the NMI read/write function that was just read from the transaction buffer is gated by the PAR MUX SEL code from the NBIM MCA (over the ENBUS) and latched in the NBFD MCA.

1. The data in the first DMA data location is loaded from the transaction buffer into the NMI data buffer. The buffer now holds both the DMA address (in its NMI transmit latches) as well as the DMA data (in its EBus receive latches).

2. The mask (if any) in the DMA data location is latched in the NBIM MCA. It is loaded in the same latches holding the NMI function, which now has been gated to (and latched in) the NBFD MCA.

This reading of the first DMA data location in the transaction buffer occurs for both a DMA read or write request. For a DMA write, its function is (of course) to read the first and possibly the only longword of write data from the transaction buffer. For a DMA read, it reads the previous contents of the transaction buffer location and has no real in the overall operation. Nothing now occurs until the NBIA wins the NMI. Then, I/O BUS EN asserts NMI ENABLE causing the following:

1. The NMI data buffer transmits the address it is holding on the NMI ADDRESS/DATA lines.

2. The NBFD MCA transmits the read/write function it is holding on the NMI FUNC lines.

3. The NBIM MCA transmits the commander's ID on the NMI ID/MASK lines. The ID is generated from the hard-wired I/O SEL signal (identifies the NBIA) and a DSEQ BI1 XACTION signal from the NPAR MCA that identifies the NBIB (VAXBI) making the DMA request. This last signal is derived from BI1 DMA IN PROGRESS, which is asserted by the DSEQ MCA when it receives a DMA request from VAXBI1. There is also a DMA IN PROGRESS signal for VAXBI0.

4. The transmit sequencer in the NBAP MCA asserts I/O HOLD on the NMI if the operation is a DMA write, and it asserts C/A XMITD. The NPAR MCA uses this last signal, together with the DMA IN PROGRESS signal asserted by the DSEQ MCA, to assert CONF DUE. (Again, there is a CONF DUE signal for each VAXBI.) Its main function is to condition the DMA response logic in the NBAP MCA. This logic responds to the confirmation sent by memory after it receives the command/address just transmitted by the NBIA.

After the DMA REQ signal is asserted by the DSEQ MCA and before the NMI is won, it is possible for the DMA request to be aborted by the receive sequencer. (The reading of the command/address and/or the contents of the first DMA data location from the transaction buffer may also be aborted.) This occurs if return read data from a previous DMA request is being sent to the NBIA by the memory. To abort the DMA request, the receive sequencer asserts an ABORT signal that deasserts I/O ARB in the transmit sequencer. The DSEQ MCA will then reinitiate the DMA request (assert DMA REQ again) when the DMA return read data has been loaded in the transaction buffer. The loading of the return read data is described in Section 3.4.3.

3.4.2 Write Data Transfer

Like the transfer of the command/address, the transfer of write data (and mask, if any) from the VAXBI to the NMI consists of three basic operations. (Refer to Figure 3-17.) However, for multilongword transfers, each operation is repeated two times (for quadword writes) or four times (for octaword writes).

1. The NBIB transfers the write data received by its BIIC into its BCI data buffer and then immediately into its data bus buffer.

2. The NBIA transfers the write data held in the NBIB's data bus buffer into its own transaction buffer.

3. The NBIA reads the write data from its transaction buffer into its NMI data buffer and then transmits it on the NMI.

3.4.2.1 Write Data to BCI Data Buffer and Data Bus Buffer -- After the command/address of a VAXBI write transaction has been received by the BIIC and moved to L6 in the NBIB's data bus buffer, the slave port sequencer moves from the WRITE WAIT state to the WRITE DATA state. In then remains in this state for as long as is necessary to move each longword of write data (and write mask) received by the BIIC to the data bus buffer.

The sequencer's WRITE DATA state transfers each longword of write data (and its mask bits) by causing the data buffer control to assert MY CLE, I TO F RD EN, and a data bus buffer latch enable.

The MY CLE signal first allows the received data and mask to be clocked into the BCI data buffer (L4 and L6). The I to F signal and the latch enable then cause the data and mask to be loaded almost immediately into the data bus buffer. During the transfer, the mask bits are passed through I to F logic unmodified with one exception. Because there is no unmasked NMI quadword transaction, the I to F logic sets the mask bits to all ones when the VAXBI quadword transaction is an ordinary write (WR or WCI). The transaction type is known because the VAXBI command and the data length were stored (in the BUF I and BUF D<31:30> latches) during the command/address transfer.

The latches that are loaded in the data bus buffer depend on which longword of data is being moved. The first longword and mask are loaded in L8 (the only latch loaded during a longword transfer). If there is a second longword (a quadword transfer), it and its mask are loaded in L1. Similarly, if there is a third and fourth (an octaword transfer), they are loaded in L2 and L6.

The last longword and mask can be loaded in L6 (which also holds the command/address) because the command/address has been taken by the NBIA by this time. This is true even during backed up DMA write operations when the NBIA is not taking the write data shortly after it is loaded in the NBIB latches. That is, even though the first operation has not completed (data not written into memory), the NBIA still takes the command/address for the second operation when it is first loaded in L6. It then waits until the first operation ends before taking the data for the second.

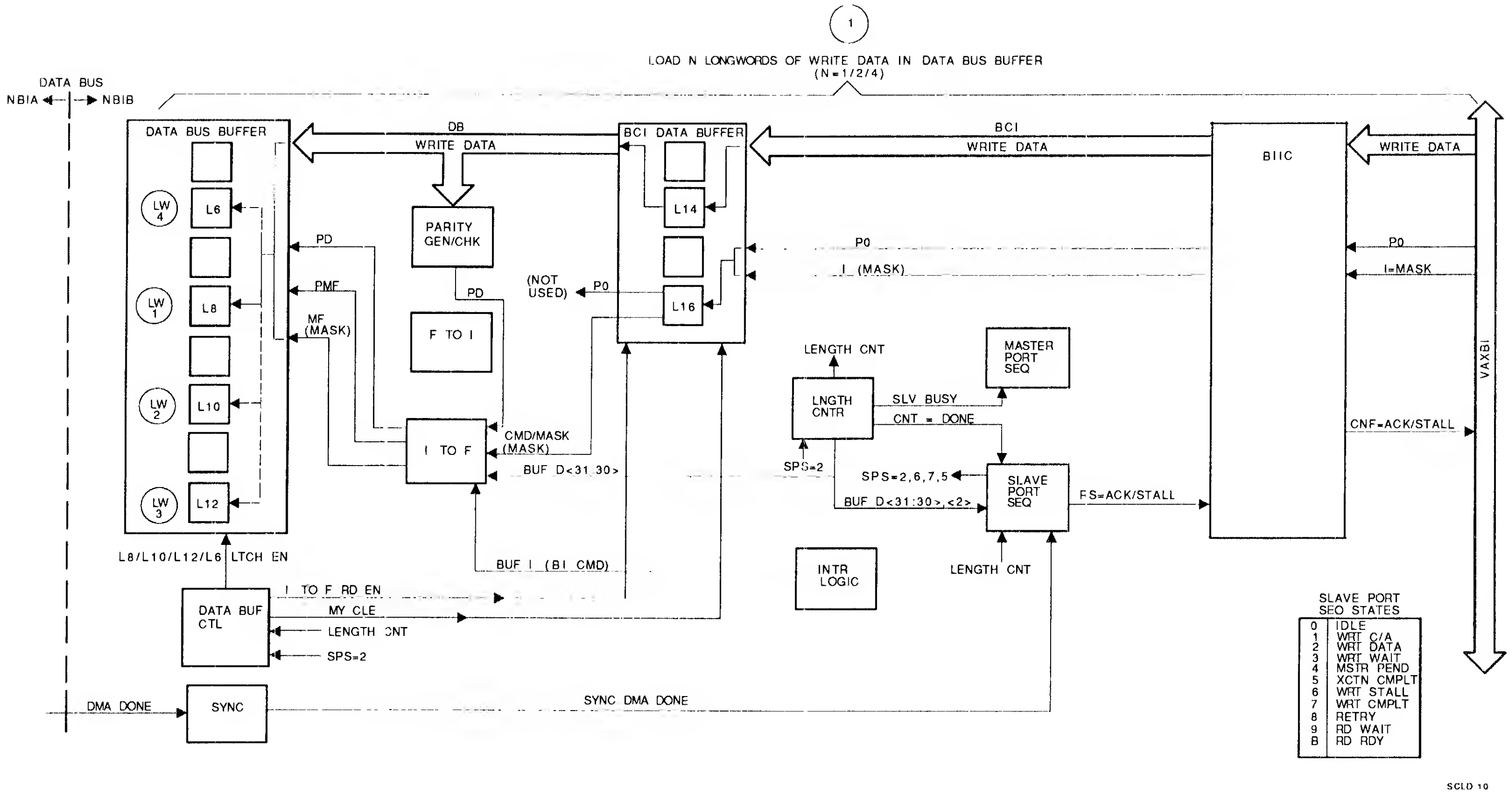
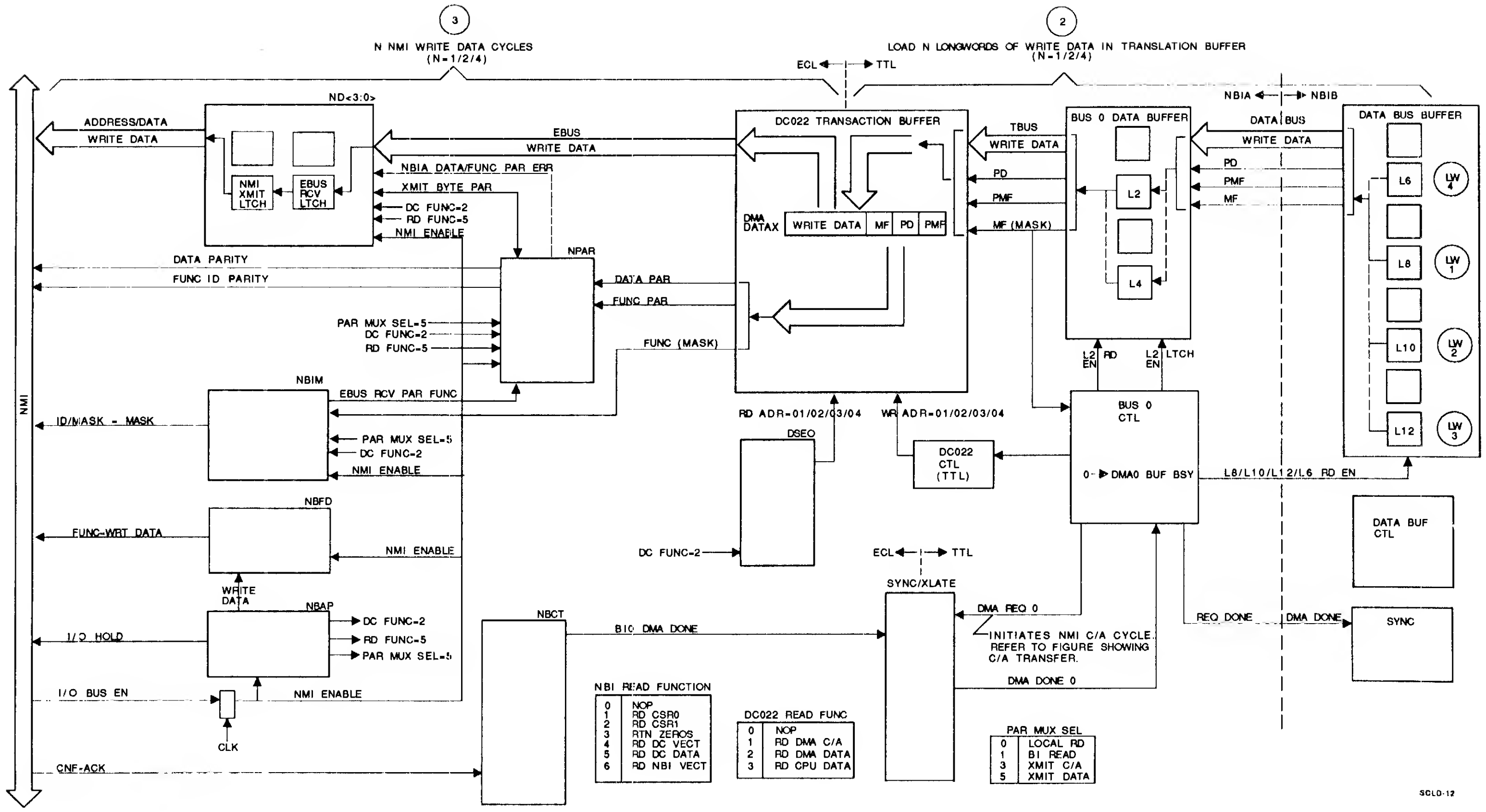


Figure 3-17 VAXBI to NMI Write Data Transfer (Sheet 1 of 2)



NBI READ FUNCTION

0	NOP
1	RD CSR0
2	RD CSR1
3	RTN ZEROS
4	RD DC VECT
5	RD DC DATA
6	RD NBI VECT

DC022 READ FUNC

0	NOP
1	RD DMA C/A
2	RD DMA DATA
3	RD CPU DATA

PAR MUX SEL

0	LOCAL RD
1	BI READ
3	XMIT C/A
5	XMIT DATA

Figure 3-17 VAXBI to NMI Write Data Transfer (Sheet 2 of 2)

3.4.2.2 End of VAXBI Transaction (and Retries) -- When the last longword of write data has been transferred, length counter output CNT = DONE is asserted. (The transaction length count was loaded in the length counter logic during the command/address transfer.) The slave port sequencer then advances to the WRITE STALL state stalling the VAXBI transaction (asserts a STALL code on the RS lines to the BIIC) until the NBIA indicates it has taken the write data. The NBIA does this by asserting REQ DONE (DMA DONE in the NBIB) which, after synchronization to the NBIB clock, causes the slave port sequencer to advance to the WRITE COMPLETE state, then to the TRANSACTION COMPLETE state, and finally to the IDLE state. During the first two states, an ACK code is asserted on the RS lines to the BIIC causing it to acknowledge (on the VAXBI CNF lines) that the transaction completed successfully. The slave port sequencer is then free to accept another DMA write (or read) transaction from the VAXBI.

As can be seen, the VAXBI write transaction is normally ended when the NBIA has taken the write data. (The data has not yet been written in memory.) However, a diagnostic control bit called FORCE DMA BUSY can be set in the NBIA (in CSR0) that holds the NBIB's slave port sequencer in the WRITE STALL state and forces the BIIC to generate a stall timeout (after 128 consecutive stall responses). When the timeout occurs, the BIIC NOACKs the node originating the transaction (ending the VAXBI transaction), and it asserts an STO event code that causes the slave port sequencer to go the IDLE state ending the DMA write operation in the NBIB.

3.4.2.3 Write Data to NBIA's Transaction Buffer -- After an NBIA's data bus control has written the command/address for a DMA write operation into the NBIA's transaction buffer, it transfers all write data from the NBIB and also writes it in the transaction buffer. The data is transferred (to either L2 or L4 in the NBIA's data buffer) and written into the transaction buffer one longword at a time. (The data bus control has its own length counter, conditioned by the NMI read/write function taken during the command/address transfer, to indicate the end of the transfer.) The data bus control then asserts a DMA REQ signal, which initiates an NMI command/address cycle as discussed previously in Section 3.4.1. The control also asserts the REQ DONE signal that ends the VAXBI transaction in the NBIB.

3.4.2.4 Write Data to NMI (NMI Write Data Cycle or Cycles) -- After the DSEQ MCA reads the command/address from the transaction buffer, it also reads the first longword of write data from the transaction buffer as described previously in Section 3.4.1. Then, when the NBIA wins the NMI, the command/address followed by the first longword of write data (and its mask) are transmitted on the NMI while the DSEQ MCA reads the next DMA data location in the transaction buffer. This ends a longword write transaction (a single NMI write data cycle following the command/address cycle), and the memory acknowledges the transaction by asserting an ACK code on the NMI CNF lines during the next bus cycle.

If the NMI transaction is a multilongword write, the transmit sequencer in the NBAP MCA again asserts a DC022 read function (READ DMA DATA) and an NBI read function (READ DC022 DATA), just as it did for the first longword of write data. As before, this causes the write data (now the second longword) to be loaded in the NMI data buffer, and its mask to be loaded in the NBIM MCA. With NMI ENABLE still asserted, the data and mask are also transmitted on the NMI (a second NMI write data cycle) while the DSEQ MCA reads the next DMA data location in the transaction buffer. This ends an NMI write quadword transaction. For an NMI write octaword transaction, the operation repeats to generate two more NMI write data cycles before the transaction ends. During any of the NMI write transactions, the I/O HOLD line (asserted by the transmit sequencer during the command/address cycle) is deserted during the last write data cycle.

When the memory acknowledges the NMI write transaction (after the first write data cycle), the NBCT MCA asserts one of two DMA DONE signals. (The signal asserted depends on the NBIB making the original DMA request.) This signal, after ECL to TTL translation and synchronization to the TTL clock, then clears the DMA BUF BSY control bit in the associated data bus control ending the DMA write operation. With DMA BUF BSY cleared, the data bus control is free to accept another DMA request from the NBIB.

3.4.2.5 NMI Write Transaction Retries (NO ACCESS/MEMORY BUSY/NOACK) -- If the NBIA won the NMI and has started an NMI write transaction, and MEMORY BUSY is asserted by memory (on the NMI) during the command/address cycle or first write data cycle, the transaction is retried. The retry is done by the NBAP's transmit sequencer, which checks the MEMORY BUSY signal after it generates the read functions and other signals that control the transmission of the command/address and first longword of write data on the NMI. If MEMORY BUSY is asserted, the sequencer stops the transmitting of write data (if there is more to transmit), allows the I/O HOLD line to be deserted (I/O BUS EN is then deserted by the bus arbitrator in the CPU), and then it requests the bus again and reinitiates the transaction just as when it first received the DMA REQ signal from the DSEQ MCA. The transaction then completes as it does normally unless another retry is required.

An NMI write transaction will also be retried if the transaction is not acknowledged by memory (usually because of a nonexistent memory address). In this case, the command/address and all write data are transmitted on the NMI as is done normally. However, when an ACK response is not received by the NBIA, the NBCT MCA asserts RETRY (BI0 or BI1) DMA REQ which causes the DESQ MCA to assert DMA REQ again. The transmit sequencer then reinitiates the NMI transaction.

3.4.2.6 DMA Errors -- A DMA write operation will be terminated if the NBIA cannot win the NMI (after a timeout), or if it is retrying the NMI transaction and the retry counter times out. When this occurs, DMA ERROR (there is a signal for each VAXBI) is asserted by the NBCT MCA. This signal, like DMA DONE, then clears DMA BUF BSY in the associated data bus control ending the DMA write operation. DMA ERROR is asserted by the following:

1. A bus access timeout (Section 3.4.5)

2. A retry timeout (Section 3.4.5)

3.4.3 Return Read Data Transfer

The transfer of return read data (and status) from the NMI to the VAXBI is shown in Figure 3-18. Like the transfer of command/address or write data/mask information in the opposite direction (from the VAXBI to the NMI), it consists of three basic operations.

1. The NBIA loads the read data/status received from the NMI into its DC022 transaction buffer.

2. The NBIA reads the read data/status from its transaction buffer and loads it in the NBIB's data bus buffer (over the interconnecting data bus).

3. The NBIB reads the read data/status from its data bus buffer into its BCI data buffer where it is quickly taken by the BIIC and transmitted on the VAXBI (a VAXBI read cycle).

Also, similar to write data/mask transfers, each operation is repeated for multilongword transfers. For octaword transfers, each is repeated four times. For VAXBI quadword transfers (octaword transfers on the NMI), the first two steps are repeated four times but the third is only repeated twice when the quadword is aligned. However, when the quadword is unaligned, the third step is also repeated four times but the BIIC only transmits read data/status on the VAXBI twice (the VAXBI transaction is stalled between the first and second VAXBI read cycles).

3.4.3.1 Return Read Data to Transaction Buffer -- When the memory receives the command/address for an NMI read transaction, it first acknowledges the transaction by asserting an ACK code on the NMI CNF lines. (The address must be valid and the memory not interlocked.) Then, when the read data has been read from the array, it requests the NMI and (after winning the bus) transmits the data in up to four consecutive data cycles to the requesting device (in this case, the NBIA). (Two of these read data transfers are required to transfer a hexword, but the NBIA generates only longword or octaword transactions.)

When the NBIA receives the ACK code from memory, it is latched in the NBCT MCA. The acknowlegdment always occurs during the second bus cycle following transmission of the command/address.

When the memory starts sending data, and during each NMI read data cycle, it transmits a longword of read data on the NMI ADDRESS/DATA lines, the commander's ID on the NMI ID/MASK lines, and either a RETURN DATA or READ CONTINUE code on the NMI FUNC lines. The RETURN DATA code is sent to identify the first longword of data. The READ CONTINUE code is sent for all others (if any). Both codes indicate if the data is good or if it is bad (cannot be corrected by the memory).

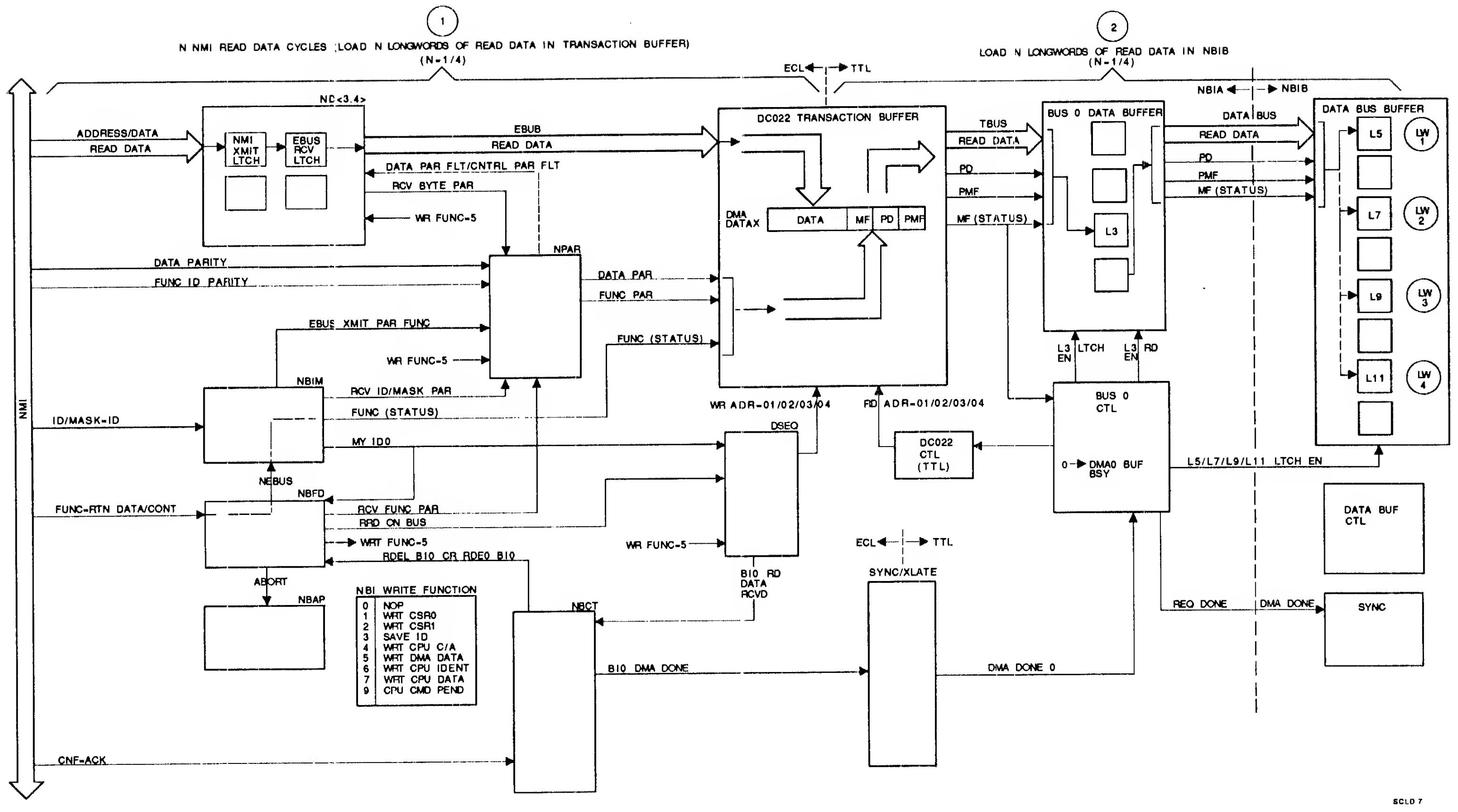


Figure 3-18 NMI to VAXBI Return Read Data Transfer (Sheet 1 of 2)

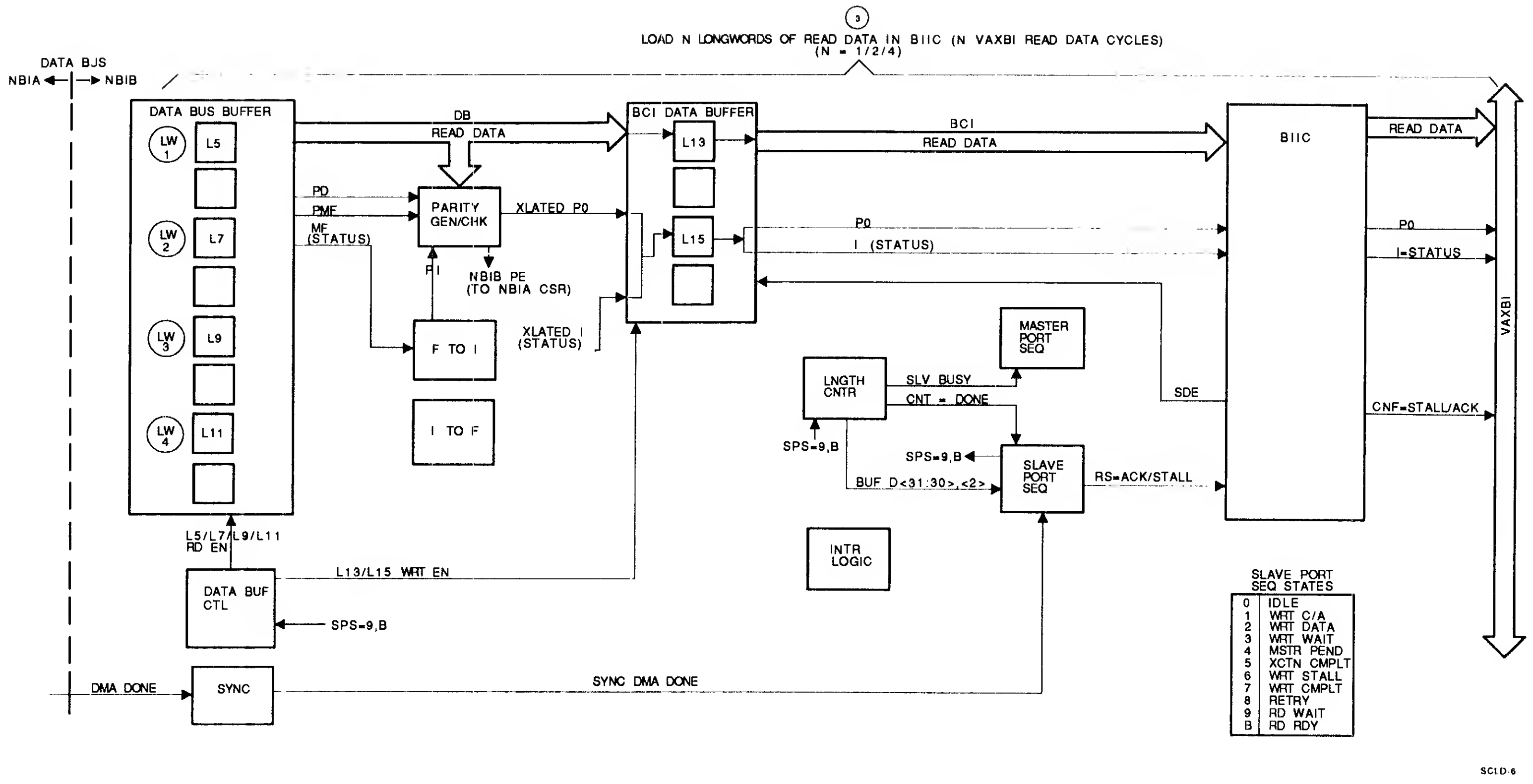


Figure 3-18 NMI to VAXBI Return Read Data Transfer (Sheet 2 of 2)

When the ID is the NBIA's ID, which also specifies the VAXBI that made the DMA request, the NBIM MCA asserts MY ID (there is a MY ID signal for each VAXBI). Also, if the NBIA is expecting read data, an RDEL or RDEO (read data expected, longword or octaword) signal will already be asserted by the NBCT MCA. This signal is asserted during the command/address transfer, and it is derived from the C/A TYPE code transmitted by the NBIM MCA. (Again, there is an RDEL or RDEO signal for each VAXBI.) The reason that there are signals for each VAXBI is that the NBIA can have two DMA read operations in progress at the same time. That is, it can send a command/address from both VAXBI0 and VAXBI1 before getting the requested read data for either.

With both a MY ID signal and a corresponding RDEL or RDEO signal asserted, and with a RETURN DATA function received on the NMI FUNC lines, the receive sequencer starts the internal transfer of the return read data by advancing state and asserting a write function (WRITE DMA DATA). It also asserts RRD ON BUS and ABORT. (The ABORT signal aborts a pending DMA write operation as described previously in Section 3.4.1.)

The write function does the following:

1. Causes the NMI data buffer to transmit the received longword of read data to the transaction buffer.

2. Causes the function code (RETURN DATA in this case) received on the NMI FUNC lines to be gated from the NBFD MCA to the NBIM MCA (over the NEBUS) and transmitted to the transaction buffer.

3. Causes the DSEQ MCA to then write the read data and received read function (the data status) into the transaction buffer.

The RRD ON BUS signal, together with a MY ID signal, causes the DSEQ MCA to assert a (BI0 or BI1) RD DATA RCVD signal. This, in turn, causes the NBCT MCA to assert (BI0 or BI1) DMA DONE.

This ends the loading of a single longword of return read data (and its status) in the transaction buffer. The data is loaded in the first of the four transaction buffer locations reserved for DMA data (for that VAXBI). If an octaword is returned from memory, the receive sequencer continues to generate the same write function, causing the remaining three longwords to be loaded in the transaction buffer as each is received on the NMI. (The status code in the transaction buffer with these longwords is the READ CONTINUE code.)

3.4.3.2 NMI Read Transaction Retries (MEMORY BUSY) -- As for an NMI write transaction, the NBIA retries an NMI read transaction if MEMORY BUSY is asserted during the command/address cycle. To do this, the transmit sequencer simply requests the NMI again. The command/address has already been read from the transaction buffer.

3.4.3.3 Return Read Data to NBIB -- The DMA DONE (0 or 1) signal, asserted by the NBCT MCA when return read data is received from the NMI, is translated from ECL to TTL, synchronized to the TTL clock, and gated to the corresponding data bus control (0 or 1). The bus control, which has been in a wait state with its DMA BUF BSY control bit set, then clears DMA BUF BSY and transfers the one or four longwords of return read data (each with its status code) to the NBIB over the interconnecting data bus. As for a CPU command/address or a CPU write data/mask transfer, a longword of return read data (and its status) is first read from the transaction buffer into the NBIA's data buffer (this time, into the L3 latches) before it is quickly passed to the NBIB's data bus buffer. Also, as when loading DMA write data from the NBIB, the counter in the NBIA's data bus control (loaded during the command/address transfer) determines when all the data has been transferred.

After transferring one longword of return read data (and its status) to the NBIB, the data bus control asserts REQ DONE. For a longword transfer, this ends the DMA read operation in the NBIA. For an octaword transfer, the DMA read operation ends when the other three longwords are read from the transaction buffer and loaded in the NBIB.

3.4.3.4 Return Read Data to BCI Data Buffer and BIIC (VAXBI READ DATA CYCLE) -- In the NBIB, REQ DONE asserts SYNC DMA DONE as it did during a DMA write operation. The slave port sequencer, currently stalling the VAXBI read transaction with the BIIC asserting SDE, then advances to the READ READY state. This causes the NBIB's data buffer control to assert the enable levels necessary to move the first longword of return read data (and its status) from the NBIB's data bus buffer (from L5) to its BCI data buffer (to L3/L5). The BIIC's SDE signal then causes the data/status to be transmitted to the BIIC where it is latched and transmitted on the VAXBI's data and I lines. The sequencer causes the data/status to be transmitted on the VAXBI (a VAXBI read data cycle) by asserting an ACK code on the BIIC's RS lines. This is the only data cycle for a VAXBI read longword transaction. If the VAXBI transaction is a read octaword transaction, the slave port sequencer stays in the READ READY state for three more bus cycles to transfer the rest of the read data/status in the data bus buffer (in L7, L9, and L1) to the BIIC and VAXBI.

For VAXBI read quadword transactions, the NBIB's data bus buffer also contains an octaword of read data (an octaword is read from memory as previously discussed), but only two VAXBI data cycles are generated. For an aligned quadword read, the slave port sequencer stays in the READ READY state long enough to transfer the first two longwords of read data in the data bus buffer (in L5 and L7) to the BIIC and VAXBI. This is done in two consecutive VAXBI data cycles. For an unaligned quadword read, the first and fourth longword of read data in the data bus buffer (in L5 and L1) are transmitted on the VAXBI. (Refer again to Figure 3-14.) To do this, and similar to an octaword transfer, the slave port sequencer stays in the READ READY state long enough to transfer all four longwords to the BCI data buffer. However, it asserts a STALL code on the RS lines to the BIIC for the second and third longwords. As a result, only two VAXBI data cycles are generated with two stall cycles occurring between the first and second.

During the internal transfer of return read data (from the data bus buffer to the BCI data buffer), the NMI status code for the longword of read data is translated to a corresponding VAXBI status code by the NBIB's F to I logic. That is, either an NMI RETURN DATA or READ CONTINUE code that indicates the data is good causes an RDDC (Read Data Don't Cache) code to be transmitted on the VAXBI. If the RETURN DATA or READ CONTINUE code indicates the data is bad, an RDSDC (Read Data Substitute Don't Cache) code is transmitted.

3.4.3.5 End of **VAXBI Transaction** -- During the last data cycle of the VAXBI transaction (longword, quadword, or octaword), the data length counter logic loaded during the command/address transfer asserts CNT=DONE. The slave port sequencer then returns to the IDLE state ending the DMA read operation in the NBIB.

After the node originating the VAXBI read transaction takes all the data from the NBIB, it asserts an ACK code on the VAXBI's CNF lines for two bus cycles unless it detects a transaction execution error (for example, a VAXBI parity error). If this ACK code is not received by the NBIB's BIIC, it sets an internal error bit and generates a VAXBI interrupt request. When the read data returned to the node originating the VAXBI read transaction is bad (an RDSDC code is transmitted by the NBIB on the VAXBI's I lines), the originating node sets an internal error bit and generates an interrupt request.

As for a VAXBI write transaction during a DMA write operation, the diagnostic control bit FORCE DMA BUSY (in CSR0) prevents a VAXBI read transaction from ending normally by forcing a stall timeout by the BIIC. Again, when the timeout occurs, the BIIC NOACKs the node originating the transaction (which ends the VAXBI transaction) and asserts an STO event code that causes the slave port sequencer to go to the IDLE state.

3.4.3.6 DMA Errors (VAXBI Transaction Retries) -- A DMA read operation is terminated, and then caused to be retried, if the command/address cannot be immediately sent to memory, or if an error is detected when return read data is received from memory. DMA ERROR, asserted by the NBCT MCA in the NBIA (there is a signal for each VAXBI), first clears DMA BUF BSY in the associated bus control. (DMA DONE normally does this.) Then, DMA ERROR causes the slave port sequencer in the associated NBIB to assert a RETRY code on the RS lines to the BIIC and go to the IDLE state. DMA ERROR is asserted by the following:

1. A bus access timeout (Section 3.4.5)

2. A retry timeout (Section 3.4.5)

3. No response from memory

4. An interlocked response from memory

5. An NMI read sequence fault (Section 3.4.6)

6. An NMI data parity fault (Section 3.4.4)

3.4.4 Parity Generation and Checking

3.4.4.1 Command/Address and Write Data/Mask Parity -- Parity for DMA command/address and write data/mask information (like CPU return read data/mask information) is checked when the information is received by the NBIB's BIIC, regenerated by the NBIB when the information is loaded in the NBIB's data bus buffer, and then checked and regenerated again by the NBIA when the information is read from the transaction buffer and transmitted on the NMI.

If the NBIB's BIIC detects a parity error for the command/address, it sets an internal error bit and generates an interrupt request. (All nodes could interrupt as they all check parity during command/address cycles.) Also, the NBIB's BIIC does not respond to the command/address and a DMA read/write operation is not started in the NBI. If the BIIC detects a parity error when taking write data/mask information, it also sets an error bit and generates an interrupt request. (Only the nodes taking and transmitting write data check its parity.) Write data/mask information having bad parity is accepted by the BIIC and passed to the NBIA and memory.

The NBIB regenerates both data and control parity bits (PD and PF) for transmission to the NBIA as follows:

1. PD - The NBIB's parity generator/checker computes even parity for the memory address and all longwords of write data when they are transferred to the NBIB's data bus buffer. The parity bit is gated to the data bus buffer (along with the PMF bit) by the NBIB's I to F logic.

2. PMF - When the VAXBI command and the write mask associated with the write data are gated through the I to F logic and loaded in the data bus buffer (the VAXBI command is translated to an NMI function and, for one VAXBI command, the mask bits are set to ones), the I to F logic computes an even parity bit (PMF) for the information and gates it to the data bus buffer along with the data parity bit (PD).

The NBIB will make both PD and PMF odd parity bits if the FORCE NBIB PE control bit in CSR1 (in the NBIA) is set. This bit is set during diagnostic operations to test the parity checking circuits in the NBIA.

In the NBIA, when the command/address and write data/mask information is read from the transaction buffer, the following occurs:

1. The NMI data buffer generates parity for each byte of address or write data, and the NBIM MCA generates an even parity bit for the NMI function or write mask.

2. The NPAR MCA, using the byte parity bits from the NMI data buffer, then generates even parity for all four bytes and checks the result with the data parity bit (PD) read from the transaction buffer. Also, it compares the parity bit generated by the NBIM MCA with the control parity bit (PMF) read from the transaction buffer. If an error is detected, NBIA DATA PAR ERR or NBIA FUNC PAR ERR sets the corresponding parity error bit (in CSR0 or CSR1) causing an NBI interrupt request to be asserted.

The NPAR MCA also generates the NMI data and control line parity transmitted during the DMA command/address and write data cycles. It uses PD from the transaction buffer (directly) to generate the data line parity, and it uses the parity bit from the NBIM MCA to generate control line parity. During write data cycles, the NBIM parity computation (for the write mask transmitted on the NMI ID/MASK lines) is valid and can be used directly because the write data code transmitted on the NMI FUNC lines has an even number of bits. However, during command/address cycles, the NBIM parity bit (for the read/write function transmitted on the NMI FUNC lines) has to be inverted if the ID transmitted on the NMI ID/MASK lines has an odd number of bits. That is, the command/address is from NBIB (from a VAXBI1 device).

3.4.4.2 Return Read Data/Status Parity -- Parity for DMA return read data/status information (like CPU command/address and CPU write/mask information) is checked by the NBIA when the information is taken from the NMI, regenerated by the NBIA when the information is written in the transaction buffer, and then checked and regenerated again in the NBIB when the information is transferred from the data bus buffer to the BIIC for transmission on the VAXBI.

During NMI return data cycles, the NMI data buffer computes an even parity bit for each of the four bytes of return read data received. Similarly, the NBIM and NBFD MCAs compute even parity bits for the received ID and read data status. The NPAR MCA then completes the data and control line parity computation comparing the results with the data and control parity bits received from the NMI. If there is a parity error, the NPAR MCA asserts DATA PAR FLT or CNTRL PAR FLT causing the corresponding error bit to be set in CSR0. This, in turn, causes the NBI's FAULT DETECT line to be asserted on the NMI, which generates an interrupt request in the CPU. (Because all NMI nexus check NMI parity during every bus cycle, all could be asserting their NMI FAULT DETECT lines.) The detection of an NMI parity error does not stop the NBIA from writing the read data and status into the transaction buffer. However, DATA PAR FLT asserts (BI0 or BI1) DMA ERROR, which causes the associated NBIB to send a RETRY response to the VAXBI node originating the VAXBI read transaction (Refer to Section 3.4.3.) DMA ERROR is not asserted by CNTRL PAR FLT. If control line information (and not just the control line parity bit) is bad, a read sequence fault will be detected and assert DMA ERROR. (If just the control parity bit is bad, the read data is transferred to the NBIB and taken by the VAXBI node as is done normally.)

The NPAR MCA also generates both the data and control (even) parity bits, PD and PMF, that are loaded in the transaction buffer along with (in this case) the read data and status. To generate PD (read data parity) and PMF (read status parity), it does the following:

1. PD - Gates the received NMI data parity bit to the transaction buffer.

2. PMF - Gates the parity bit computed by the NBIM MCA to the transaction buffer. This is not the parity bit for the ID received from the NMI. It is the parity bit computed for the read status code when the code is passed through the NBIM MCA (from the NBFD MCA) to the transaction buffer.

Data and control line parity are checked in the NBIB when the return read data and status are transferred from the data bus buffer to the BCI data buffer. If a parity error is detected, the NBIB's parity generator/checker asserts NBIB PE. Again, a parity error does not stop the operation. (The read data/status is transferred to the BIIC and VAXBI.) However, NBIB PE sets an error bit in the NBIA (in CSR0) causing an NBI interrupt request. Diagnostics can force the error signal to be asserted by setting a control bit (FORCE NBIB PE) in CSR1.

The parity generator/checker also generates the single VAXBI odd parity bit (P0). (The NBIB's BIIC is not enabled to generate parity on the VAXBI.) The bit is loaded in the BCI data buffer, passed to the BIIC, and transmitted on the VAXBI along with the read data and status. It is generated from the data parity bit (PD) sent by the NBIA, and a parity bit (PI) for the read status that is computed by the NBIB's I to F logic. The control parity bit (PMF) from the NBIA cannot be used because the I to F logic translates the NMI read status code (from the NBIA) to a VAXBI status code.

When read data and status are transmitted on the VAXBI, both the NBIB's BIIC and the BIIC in the node taking the data/status check for parity errors. If an error is detected, each sets an internal error bit and generates an interrupt request.

3.4.5 Timeouts

There are two timeout counters in the NBIA. One, the bus access timeout counter in the NBAP MCA, asserts BUS ACCESS TMOUT if the transmit sequencer (after requesting the NMI) does not win the bus after two system slow clock periods. This is approximately six milliseconds at the normal system clock rate. The other counter is the retry timeout counter in the NPAR MCA. It asserts (BI0 or BI1) TIMEOUT if the NMI transaction continues to be retried by the transmit sequencer for (again) a period of two system slow clock periods or approximately six milliseconds. The counter is started when the transmit sequencer first wins the bus and transmits the command/address to memory (DMA C/A XMTD = 1). A timeout can then occur because MEMORY BUSY continues to be asserted, or because the memory continues to return no response or an interlocked response.

After a bus access or retry timeout occurs, the NBCT MCA asserts DMA ERROR and loads the appropriate error code in its timeout encoder. For retry timeouts, the code is generated from the last type of response received from memory (no response, interlocked, or memory busy). The timeout code asserted is loaded in CSR0, and causes an NBI interrupt request to be generated.

3.4.6 Read Sequence Faults

The receive sequencer in the NBFD MCA asserts RD SEQ FLT (which asserts DMA ERROR) if the following occurs:

1. Return read data is sent to the NBIA when it is not expecting it. That is, read data is sent with a RETURN DATA code on the NMI FUNC lines (which identifies the first longword in a transfer), but no corresponding RDEL or RDEO (Read Data Expected, Longword or Octaword) signal is asserted. An RDEL or RDEO signal is asserted by the NBCT MCA at the beginning of every normal DMA read operation after the command/address is transmitted to memory.

2. Not enough return read data is sent by memory after it sends the first longword to the NBIA (incomplete read data). That is, a READ CONTINUE code is not received on the NMI FUNC lines during a bus cycle when more read data is expected. (The READ CONTINUE code identifies all longwords of read data following the first.)

3.5 INTERRUPT (INTR AND IPINTR) OPERATIONS

The NBIB is the device interrupt fielding node on its associated VAXBI. In this capacity, it responds to all VAXBI INTR transactions directed to it by causing the NBIA to generate an interrupt request on the NMI. The CPU then responds to the request by reading an interrupt vector from the interrupting VAXBI node. (The NBIB generates a VAXBI IDENT transaction when the CPU reads an NBIA vector register address.)

The NBIB also responds to interprocessor interrupt requests (VAXBI IPINTR transactions) from I/O processor nodes on the VAXBI. Furthermore, it can generate a VAXBI interprocessor interrupt request. The IPINTR transaction is generated when the CPU sets a control bit in the BIIC's BCI control register.

Device interrupt requests directed to the NBIB can be generated by its own BIIC. This is how the NBIB's BIIC normally flags any errors it detects. It is also the means by which the NBIB fields an interprocessor interrupt. When the IPINTR transaction is decoded in the NBIB, the NBIB forces a standard device interrupt request by its BIIC, which is then serviced by the CPU like any other interrupt.

The fielding of interrupt requests consists of two basic operations.

1. Decoding the VAXBI INTR or IPINTR transaction.

2. Generating the NMI interrupt request (for an INTR transaction) or the VAXBI interrupt request (for an IPINTR transaction).

Refer to Figure 3-19.

3.5.1 Decoding Interrupt Requests

When the BIIC detects a command/address on the VAXBI, it asserts CLE and transmits the command/address on the BCI I lines and data lines. The CLE signal then causes the command/address to be loaded in the BCI data buffer, and it also causes the VAXBI command (on the I lines) and the states of data lines <19:16> to be latched in the NBIB's interrupt logic. This occurs for all command/address cycles on the VAXBI. The states of the four data lines are latched because they specify the interrupt request level(s) for INTR commands. More than one data line can be asserted, allowing a node to make interrupt requests at more than one level with a single INTR transaction.

D<19:16>	Interrupt Request Level	Priority
1XXX	BR7	Highest
X1XX	BR6	
XX1X	BR5	
XXX1	BR4	Lowest

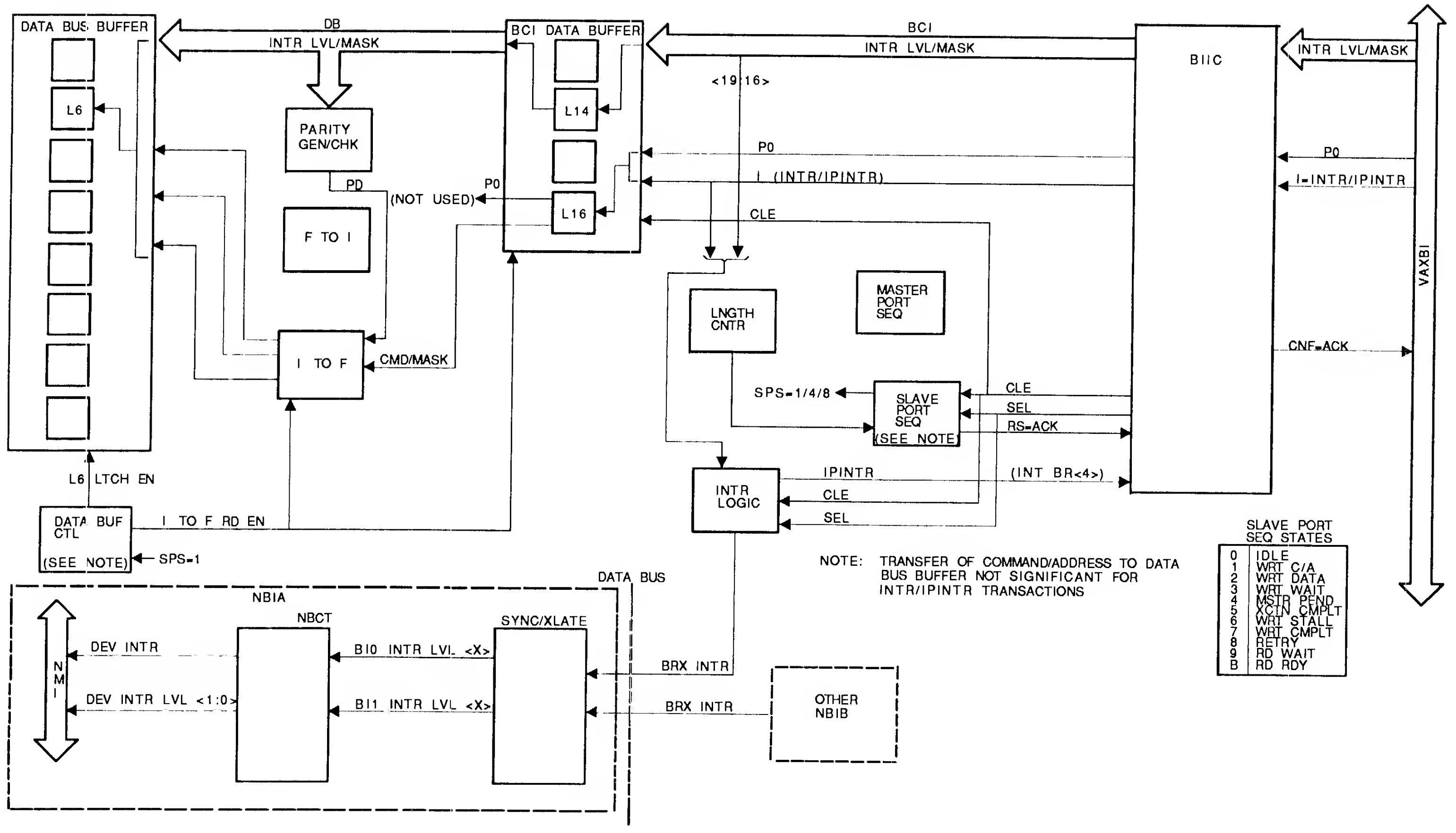


Figure 3-19 INTR/IPINTR Operations

The slave port sequencer can be in one of three states (WRITE C/A, MASTER PENDING, or RETRY) when CLE is asserted and the VAXBI command/address is loaded in the BCI data buffer and interrupt logic. If in (or if advanced to) the WRITE C/A state when CLE occurs, the command/address is also loaded from the BCI data buffer to the data bus buffer. This is because the sequencer can begin a DMA read/write in this state, and the command/address (for a VAXBI read/write transaction) must be loaded in the data bus buffer where it can be taken by the NBIA. However, this transfer has no significance when the command/address is for an INTR or IPINTR transaction.

When the command/address for a VAXBI transaction is directed to the NBIB, the BIIC asserts SEL after it asserts CLE. For example, a VAXBI read/write transaction causes SEL to be asserted when the address (a memory address) is within the range specified by the BIIC's starting and ending address registers. For VAXBI INTR transactions, SEL is asserted when the appropriate interrupt destination mask bit in the command/address is equal to one. For IPINTR transactions, the appropriate destination mask bit must also be equal to one with the additional requirement that the corresponding bit in the BIIC's IPINTR mask register must be set. The BIIC may be prevented from asserting SEL for either type of interrupt request if normally set enable bits in the BIIC's BCI control register are cleared.

The following occurs when SEL is asserted for an INTR transaction.

1. The NBIB's interrupt logic uses the interrupt request level(s) asserted on data lines <19:16>, and latched by CLE, to assert interrupt request(s) to the NBIA. A request remains asserted until cleared by the IDENT transaction that reads the corresponding vector.

Latched D<19:16>	Interrupt Request(s) to NBIA
1XXX	BR7 INTR
X1XX	BR6 INTR
XX1X	BR5 INTR
XXX1	BR4 INTR

2. The slave port sequencer asserts an ACK code on the RS lines to the BIIC. The BIIC then transmits the ACK code on the VAXBI's CNF lines ending the VAXBI transaction.

3. In the NBIA, an interrupt request from the NBIB is first synchronized to the TTL clock and then translated from a TTL to ECL level. Then, if the NBI INTERRUPT ENABLE bit is set in CSR0, the interrupt request from the NBIB causes the NBCT MCA to assert an interrupt request on the NMI if another higher priority request is not asserted. The higher priority request can be from the other NBIB in the NBI configuration. To make an NMI interrupt request, the NBCT MCA asserts DEV INTR on the bus. It also asserts the request level, encoded as a 2-bit binary number, on the DEV INTR LVL lines.

When SEL is asserted for an IPINTR transaction, the NBIB's interrupt logic asserts a signal (IPINTR) connecting to one of the BIIC's four interrupt request inputs (BCI INT <7:4>). The input asserted (BCI INT <4>) requests the BIIC to interrupt at a BR4 level. Also, as for an INTR transaction, SEL causes the slave port sequencer to assert an ACK code on the BIIC's RS inputs. The BIIC then transmits an ACK code on the VAXBI ending the IPINTR transaction.

With its BCI INT<4> input asserted, and following the IPINTR transaction, the BIIC arbitrates for the VAXBI and initiates an INTR transaction (to itself). As just described, the INTR transaction causes the NBIB to assert an interrupt request to the NBIA. (In this case, BR4 INTR is asserted). The NBIA then asserts an interrupt request on the NMI.

3.6 MISCELLANEOUS OPERATIONS

The registers in the NBIB's BIIC can be accessed by other VAXBI nodes. Also, the NBIB's BIIC can generate VAXBI STOP transactions in addition to the read/write, IDENT, and interrupt transactions already discussed.

3.6.1 BIIC Register Read/Write Operations (by Other VAXBI Nodes)

The registers in the NBIB's BIIC are normally read and written by the CPU. The registers are addressed like the registers in any other node on a VAXBI. However, the BIIC registers can also be accessed by one of the VAXBI devices (an I/O processor node, for example.) During one of these (local) VAXBI read/write operations, and during a CPU read/write operation to a BIIC register for that matter, the BIIC responds to the VAXBI read/write transaction independent of any control by the NBIB's slave port sequencer. That is, the response to a BIIC register read/write is automatic by the BIIC. For this reason, the control bit in the BIIC's BCI control register enabling the assertion of SEL (to the sequencer) during BIIC register read/write transactions is normally cleared.

3.6.2 VAXBI Stop Transactions

The CPU can stop all activity on a VAXBI, usually after an error condition has been detected, by causing the BIIC in the associated NBIB to initiate a VAXBI STOP transaction. It does this by setting a control bit in the BIIC's BCI control register. Because the control bit is the same one used to initiate an IPINTR transaction, another control bit must first be cleared in the BIIC's IPINTR/STOP command register. The STOP command code must be loaded in this register as well, and the BIIC's IPINTR/STOP destination register must be loaded to specify the nodes to be stopped.

When a VAXBI node's BIIC is selected by a STOP transaction, it is forced to a state in which it cannot issue any more VAXBI transactions. However, it can respond to VAXBI read transactions so that error status bits may be examined. A node can be returned to normal operation by forcing a selftest operation.

3.6.3 VAXBI INVAL and BROADCAST Transactions

An NBIB does not respond to VAXBI INVAL and BROADCAST transactions. Although the NBIB's BIIC may be selected, the BIIC does not assert SEL and, thus, the slave port sequencer ignores the transaction. The SEL line is not asserted because the enabling control bits in the BIIC's BCI control register are normally cleared.

3.7 DIAGNOSTIC DATA TRANSFERS

There are three types of NBI diagnostic data transfers.

1. BIIC loopback requests
2. NBIA wraparound
3. CPU read/write to memory (FLIP 29/22)

3.7.1 BIIC Loopback Requests

CPU read/write operations can be executed that do not require the use of the VAXBI data lines. When the BIIC LOOPBACK control bit is set in the NBIA (in CSR0), and a CPU read/write to VAXBI address space is initiated, the master port sequencer in the NBIB does not make a normal VAXBI transaction request to the BIIC. It makes a loopback request instead.

BIIC operation during a loopback request is as follows:

1. The BIIC ignores all but bits <12:00> of the VAXBI address transferred from the NBIA. Thus, only registers within the BIIC's own nodal address space (its internal registers) can be addressed.

2. Also, the BIIC does not arbitrate for the bus or generate a VAXBI read/write transaction as it does normally. The transfer of BIIC register read/write data is made via an internal data path.

BIIC loopback requests are used to isolate the VAXBI during diagnostic operations. They also have a function during normal operation.

At powerup, the node ID is loaded automatically in the BIIC's BCI control/status register as described in Section 3.2. This value, which depends upon which ID plug is inserted in the backplane, cannot be determined by the CPU by means of a normal CPU read operation. (The CPU needs the node ID to access a BIIC register normally.) Consequently, the CPU makes a BIIC loopback request to read the ID.

3.7.2 NBIA Wraparound

If a CPU read operation to a VAXBI address is executed with control bit NBIA WRAPAROUND set (in CSR1), the read command/address is passed through the NBIA and transmitted on one of data bus ports as is done normally. However, the command/address (now in the corresponding NBIA data bus buffer) is passed back through the NBIA and the address returned to the CPU as read data. This allows a large portion of the NBIA's data path and control circuits to be checked without the need for an NBIB.

The normal transfer of the read command/address through the NBIA is as described in Section 3.3.3.1, and as shown in Figure 3-11 (Sheet 1). The command/address is taken from the NMI and written into the transaction buffer. It is then read from the transaction buffer and loaded into a set of latches (L1) in a data bus buffer. The same thing occurs in wraparound mode. The only difference in overall operation is that after the command/address is loaded in the data bus buffer, a CPU REQ signal is not asserted on the data bus port (in case an NBIB is connected).

To begin the loopback of the command/address through the NBIA, the data bus control immediately asserts the latch and read enables for another set of latches in the data bus buffer (L4). These are the latches normally loaded with return read data from the NBIB. In this case, however, the latches are loaded with the command/address because it is still being transmitted on the data bus port from the other set of latches (L1) in the buffer. Once the command/address has been turned around in the data bus buffer, it is passed back through the NBIA just like return read data. That is, it is first written into the transaction buffer. Then, it is read from the transaction buffer and (in this case) the address is transmitted on the NMI data lines after the NBIA requests and wins the bus. The transfer of return read data is described in Section 3.3.3.3 and shown in Figure 3-13 (Sheet 2).

3.7.3 CPU Read/Write to Memory (Flip Address Bits <29> and <22>)

Normally, a CPU read/write (longword) operation to a VAXBI address can only access a register in a VAXBI node. The register address, a VAXBI I/O address, has address bit <29> equal to one. However, if a CPU read/write is initiated with control bit FLIP 29/22 set in the NBIA (in CSR0), the NBIA changes address bit <29> from a one to a zero after the command/address is taken from the NMI. (Another address bit, bit <22>, is also changed in value to maintain correct parity.) This makes the I/O address a memory address (bit <29> = 0). Thus, when the NBIB's BIIC generates the VAXBI read/write transaction transmitting the converted address on the VAXBI, the BIIC will respond (to its own VAXBI transaction) as it normally does to a memory reference by a VAXBI device. In other words, the CPU read/write operation causes a DMA read/write operation that transfers read/write data to/from memory instead of to/from a VAXBI node.

The flipping (changing the value) of address bits <29> and <22> allows diagnostics to loop a command/address and a longword of read/write data through the NBI without the need for a VAXBI I/O device. Furthermore, with a CPU read/write and a DMA read/write being executed concurrently, the majority of the NBI's data path and control circuitry can be checked in one operation. The two address bits are changed in value in the NBIA's NMI data buffer just before they are loaded in the transaction buffer along with the rest of the command/address.

Command/address and data flow during the diagnostic data transfer is shown in Figure 3-20. Flow is shown for both a CPU read operation and a CPU write operation.

For a CPU write operation, the NBIA takes the command/address and write data from the NMI and passes it to the NBIB causing a VAXBI write transaction to be initiated as in normal operation. Then, with address bit <29> = 0, the VAXBI transaction causes a DMA write operation to be initiated in the NBIB while the CPU write operation is still executing. That is, when the NBIB transmits the command/address and write data on the VAXBI, it also takes the command/address and write data from the VAXBI and passes it back to the NBIA. The NBIA then generates an NMI write transaction, and the command/address and write data are transferred to memory to complete the operation.

For a CPU read operation, and similar to a CPU write, the NBIA takes the command/address from the NMI and passes it to the NBIB. The NBIB then initiates a VAXBI transaction (this time, a VAXBI read transaction). Also, with address bit <29> = 0, the VAXBI transaction causes a DMA operation (this time, a DMA read operation) to be initiated in the NBIB. That is, the command/address transmitted on the VAXBI is looped back to the NBIA and an NMI transaction (this time, a read transaction) is initiated. When the read data is returned from memory, it is passed to the NBIB and transmitted on the VAXBI as in normal operation. Then, with the CPU read still executing, the NBIB takes the read data from the VAXBI and loops it back to the NBIA. The data is then transmitted on the NMI to the CPU completing the operation.

The reason a CPU read/write to a memory address can be implemented in the NBI is because the NBIB's master port sequencer (which initiates VAXBI transactions) and its slave port sequencer (which responds to VAXBI transactions) operate almost completely independent of each other. However, an exception to normal operation is that when the VAXBI read/write transaction is initiated by the master port sequencer, it always has to be retried one time.

The retry occurs because the slave port sequencer is placed in the MASTER PENDING state by a CPU read/write. However, during the first try of the transaction, the master port sequencer's READ CYCLE or WRITE CYCLE state clears SYNC REQ PENDING, which returns the slave port sequencer to the IDLE state and allows it to acknowledge the transaction when it is tried a second time. (This is the normal mechanism that allows the NBIB to perform DMA read/write operations during the time a CPU read/write is being retried.) The transaction can now be acknowledged because the command/address (and the longword of write data if a CPU write) is buffered in the BIIC during the first try. That is, the NBIB data path is now free to transfer the command/address and data for the upcoming DMA transfer.

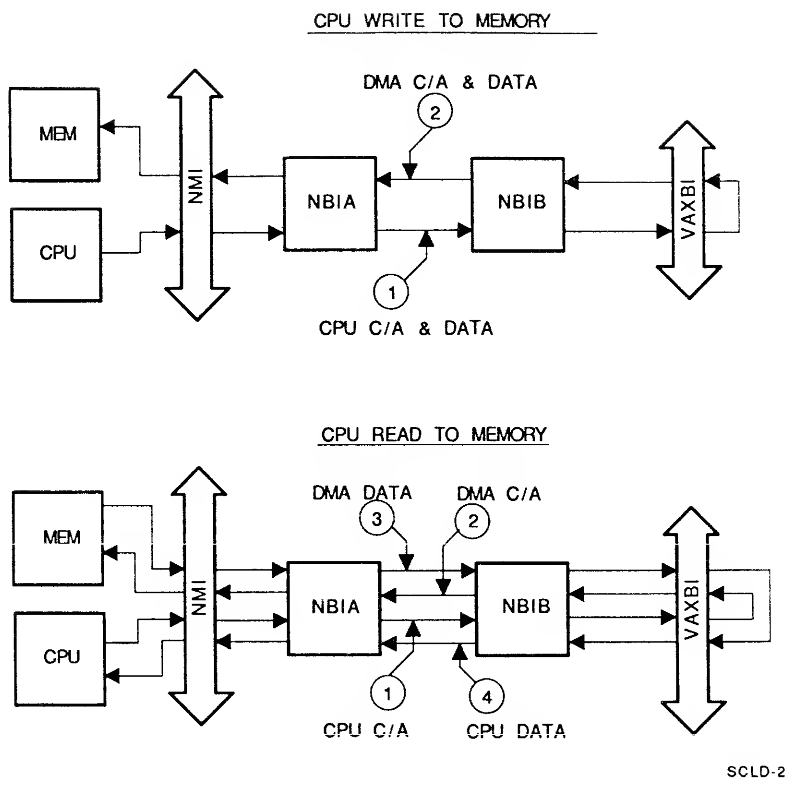


Figure 3-20 FLIP 29/22 Diagnostic Data Transfers

Digital Equipment Corporation • Bedford, MA 01730